IA Ciega 0.9 Manual de referencia

Roberto Oropeza Gamarra

Generado por Doxygen 1.4.6-NO

Tue Aug 22 12:21:37 2006

Índice

1. Documentación de IA Ciega

El objetivo de esta libreria es facilitar el uso de los Algoritmos de Búsqueda usados en Inteligencia Artificial de manera que una vez que se tengan definidos los estados y las operaciones sea sencillo probar los distintos algoritmos sin tener que implementarlos.

1.1. Introducción

La librería esta diseñada para permitir un uso sencillo sin sacrificar la flexibilidad. Se han usado plantillas, pero para llamar a las funciones no es necesario especificar los tipos de las plantillas, excepto en `primero_mejor`.

La corrida debería ser muy rápida porque siempre que es posible uso parámetros por referencia, ademas, me apoyo en la STL y plantillas para los algorítmos y contenedores necesarios.

1.2. Modo de uso

Para traer todas las funciones al scope de trabajo, evitando poner ia::... a cada rato, puedes poner:

```
using namespace ia;
```

al principio de tu programa, después de incluir las cabeceras que necesites.

Para una visión general de la libreria ve las pestañas Páginas relacionadas, Módulos, y Ejemplos, Cuando estes dispuesto a usar la librería mirá el Uso general y luego alguno de los Algoritmos de búsqueda en espacios de estados, tambien mirá los Consejos prácticos. Recuerda ver los ejemplos para ver el uso práctico y ante cualquier duda, el más sencillo es el pastor.cpp, los mas complejos el 8puzzle.cpp y reinas.cpp.

1.3. Compatibilidad y compilación

Esta librería consiste solo de un archivo cabecera: ia_ciega.h, así que no requiere compilación de librería. Se usa intensivamente la STL y plantillas, pero no se usa meta-programación, así que cualquier C++ contemporaneo debería ser suficiente. Se ha probado con éxito usando:

- Bloodshed Dev-Cpp 4.9.9.2 [=g++ (GCC) 3.4.2 (mingw-special)]
- Borland CBuilder 6 Update Pack 4 [=Borland C++ 5.6.4]

- Borland CBuilder 2006 Update Pack 2 [=Borland C++ 5.8.2]
- Borland free C++Builder 5 Command Line Tools [=Borland C++ 5.5.1]
- Digital Mars 8.4.8 [=Digital Mars Compiler 8.42n]
- Microsoft C++ de VisualStudio .NET 2003 y Visual C++ Toolkit 2003 [=Microsoft C/C++ 13.10.3077]
- Microsoft C++ de VisualStudio 2005 [=Microsoft 32-bit C/C++ Optimizing Compiler 14.00.50727.42 for 80x86]

El uso con los compilador de Microsoft Visual C++ Toolkit/VisualStudio .NET 2003 es posible solamente si se especifican los tipos de las plantillas al momento de llamar a las funciones, como se ve en los ejemplos. Esta incompatibilidad ha sido solucionada en MS VS 2005.

En el caso del compilador gratuito (MS VC++ Toolkit) no será posible usar el cronometraje automático porque tal herramienta no incluye windows.h.

No se puede usar con:

- Borland C++ 7.0 para DOS
- DJGPP

Son todos los compiladores con los que se ha probado, si alguno no esta en la lista significa que todavía no se ha probado y que podría funcionar.

1.4. Instalación

La instalación es solamente copiar el archivo ia_ciega.h a una carpeta donde la encuentre el compilador, esa carpeta suele llamarse "include".

1.5. Características y Limitaciones

- **Reentrancia:** En una aplicacion con más de un hilo debe evitar que el *mismo tipo de búsqueda (o sea la misma función de búsqueda con los mismos parámetros de plantilla)* este siendo realizado por dos hilos a la vez, debe considerarse también el mismo tipo de búsqueda a profundidad_limitada y profundidad_iterativa. Si se busca con las mismas funciones pero con diferentes parametros de plantilla no hay problemas de reentrancia.
- La capacidad de cronometrar la ejecucion solo es posible en plataforma MS Windows, en otras debe deshabilitarse con #define IA_NO_CRONOMETRAR
- Actualmente solo se ha compilado en plataforma MS Widnows, pero debe poderse en otras ya que no se usan características específicas de ese sistema (salvo en el cronometraje).

1.6. Pendientes

- Crear contenedores con características adicionales para esta libreria, como ser secuencias con allocators para memoria paginada en disco (con eso sería posible su uso en búsquedas en espacios inmensos)
- Hacer una comparación estadística práctica con otras librerias de búsqueda, tanto en tiempo como en memoria.
- Implementar otros algorítmos de búsqueda

2. IA Ciega 0.9 Documentación de módulos

2.1. Algoritmos de búsqueda en espacios de estados

Funciones

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::preferencia_amplitud (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de preferencia por amplitud.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::preferencia_profundidad (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Función para buscar una o más soluciones usando el método de preferencia por profundidad (depth-first-search).

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::profundidad_limitada (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad limitada.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::profundidad_iterativa (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad iterativa.

- template<typename Comparador_mejor_t, typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::primero_mejor (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando el algoritmo primero mejor.

- template<typename Estado_t, typename Operaciones_t> pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > ia::profundidad_limitada_doble (const Operaciones_t &operaciones, const Estado_t &inicial, const Estado_t &final, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando un algoritmo similar al de profundidad limitada, pero empezando desde el estado inicial y el estado final.

2.1.1. Documentación de las funciones

2.1.1.1. template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type> ia::preferencia_amplitud (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Funcion para buscar una o mas soluciones usando el método de preferencia por amplitud.

Parámetros:

operaciones Son todas las operaciones que se pueden aplicar para explorar el espacio de estados

inicial Es el estado inicial desde el que se empieza la búsqueda

registrar_solucion Función callback que se llamará cada que se encuentre una solucion (opcional) Uso del parámetro registrar_solucion

Devuelve:

un vector con las operaciones de la ruta a la solucion, pero si no se halló solución devuelve el vector vacio. OJO: tambien devuelve vacio si el estado inicial es el estado meta, sin embargo el callback registrar_solucion si sera llamado aunque el estado inicial sea el estado meta,

Definición en la línea 709 del archivo ia_ciega.h.

2.1.1.2. template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type> ia::preferencia_profundidad (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, bool(*)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Función para buscar una o más soluciones usando el método de preferencia por profundidad (depth-first-search).

Parámetros:

operaciones Son todas las operaciones que se pueden aplicar para explorar el espacio de estados

inicial Es el estado inicial desde el que se empieza la búsqueda

registrar_solucion Función que se llamará cuando se encuentre una solucion (opcional) Uso del parámetro registrar_solucion

Devuelve:

un vector con las operaciones de la ruta a la solucion, pero si no se halló solución devuelve el vector vacio (OJO: también devuelve vacio si el estado inicial es el estado meta).

Definición en la línea 744 del archivo ia_ciega.h.

2.1.1.3. template<typename Comparador_mejor_t, typename Estado_t, typename Operaciones_t> vector<typename Operaciones_t::value_type> ia::primero_mejor (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, bool(*)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Busca en el espacio de estados usando el algoritmo primero mejor.

Es necesario dar el primer parámetro de la plantilla, debe ser el tipo de un functor con un `operator()`, ese tipo podría ser así:

```
struct Elige_mejor {
  bool operator()( const pair< Estado, vector<Op> >& izq, const pair< Estado, vector<Op> >& der ) {
    if ( izq. ... der. ... )
         return true;
    else
         return false;
  }
};
```

El método `operator()` debe devolver true ssi el estado representado por el parámetro izq debe ser explorado antes que el estado representado por der, además debe ser un strict weak ordering como lo define la STL, con la salvedad que *no es necesario que si no(a < b) y no(b < a) entonces a y b representan el mismo estado*. Los parametros izq y der son `std::pair` cuyo miembro `first` es el estado y el miembro `second` es la secuencia de operaciones realizadas al estado inicial para llegar al estado en cuestión, Dado el functor definido más arriba, se podría llamar a la funcion así:

```
primero_mejor<Elige_mejor>( operaciones, inicial );
```

Parámetros:

operaciones Es una secuencia con todas las operaciones que se pueden aplicar a los estados

inicial Es el estado inicial desde el que se empieza la búsqueda

registrar_solucion Funcion que se llama cada que se encuentra una solucion (opcional) Uso del parámetro registrar_solucion

Devuelve:

Las operaciones aplicadas para llegar al estado meta encontrado.

Definición en la línea 860 del archivo ia_ciega.h.

2.1.1.4. template<typename Estado_t, typename Operaciones_t> vector<typename Operaciones_t::value_type> ia::profundidad_iterativa (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, const unsigned int & *limite_profundidad*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Funcion para buscar una o mas soluciones usando el método de profundidad iterativa.

Parámetros:

operaciones Son todas las operaciones que se pueden aplicar para explorar el espacio de estados

inicial Es el estado inicial desde el que se empieza la búsqueda

limite_profundidad Es la profundidad máxima a la que se explorará, iterando desde 1 hasta limite_profundidad

registrar_solucion Funcion que se llama cada que se encuentre una solucion (opcional) Uso del parámetro registrar_solucion

Devuelve:

un vector con las operaciones de la ruta a la solucion, pero si no se halló solución devuelve el vector vacio (OJO: tambien devuelve vacio si el estado inicial es el estado meta),

Ejemplos:

cantaros.cpp, y pastor.cpp.

Definición en la línea 809 del archivo ia_ciega.h.

2.1.1.5. template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type> ia::profundidad_limitada (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, const unsigned int & *limite_profundidad*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Funcion para buscar una o mas soluciones usando el método de profundidad limitada.

Parámetros:

operaciones Son todas las operaciones que se pueden aplicar para explorar el espacio de estados

inicial Es el estado inicial desde el que se empieza la búsqueda

limite_profundidad Es la profundidad máxima a la que puede llegar el árbol de búsqueda

registrar_solucion Funcion que se llamará cuando se encuentre una solucion (opcional) Uso del parámetro registrar_solucion

Devuelve:

un vector con las operaciones de la ruta a la solucion, pero si no se halló solución devuelve el vector vacio (OJO: tambien devuelve vacio si el estado inicial es el estado meta),

Definición en la línea 778 del archivo ia_ciega.h.

2.1.1.6. template<typename Estado_t, typename Operaciones_t> pair< vector<typename Operaciones_t::value_type>, vector<typename Operaciones_t::value_type> > ia::profundidad_limitada_doble (const Operaciones_t & *operaciones*, const Estado_t & *inicial*, const Estado_t & *final*, const unsigned int & *limite_profundidad*, bool(∗)(vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion* = NULL)

Busca en el espacio de estados usando un algoritmo similar al de profundidad limitada, pero empezando desde el estado inicial y el estado final.

Este no busca cualquier estado que sea meta (no usa `es_meta()`), sino el estado 'final' dado, para eso empieza la búsqueda por los dos lados (inicial y final) y considera la solución hallada a la secuencia de operaciones realizadas para llegar de inicial a un estado intermedio X y desde 'final' al mismo estado intermedio X.

Devuelve:

Un `std::pair` de rutas, la primera (`first`) consiste en las operaciones que se emplearon desde 'inicial' a un estado intermedio X, la segunda (`second`) consiste en las operaciones que se emplearon para llegar al mismo estado X, pero partiendo desde 'final'

Parámetros:

operaciones Es una secuencia con todas las operaciones que se pueden aplicar a los estados

inicial Es el estado desde el que se buscará al estado final

final Es el estado final buscado

limite_profundidad Es la profundidad máxima a la que puede llegar el árbol de búsqueda total

registrar_solucion Función que se llama cada que se encuentra una solución (opcional) Uso del parámetro registrar_solucion

Definición en la línea 899 del archivo ia_ciega.h.

2.2. Útiles para facilitar el uso

Clases

- struct ia::Operacion< Estado_t >

  Las operaciones pueden ser functores (objeto-funcion) que heredan de este, dándole como parámetro de plantilla el tipo del estado con el que opera.

- struct ia::Operaciones< Estado_t >

  La lista de operaciones se puede declarar como una instancia de esta estructura, que es un vector de punteros a las funciones que sirven de operadores.

- struct ia::OperadoresInversos< Operaciones_t >

  Si utiliza las búsquedas bi-direccionales con un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, puede utilizar este mapa para establecer las operaciones inversas.

- class ia::SecuenciaEstados< Estado_t, Operaciones_t >

  Un contenedor que se construye en base al estado inicial y a una secuencia de operadores como las que devuelven las búsquedas; pero que al iterar devuelve los estados *generados al aplicar esas operaciones.*

2.3. Útiles para mostrar resultados

Clases

- struct ia::NombresOperadores< Estado_t >

  Si se tiene un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, debería instanciar una estructura como esta, dando como parámetro de plantilla el tipo de estado con el que opera.

Funciones

- void ia::mostrar_estadisticas ()

  Muestra las estadísticas de la última búsqueda realizada.

- template<typename Estado_t, typename Operaciones_t> void ia::mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son objetos función.

- template<typename Operaciones_t> void ia::mostrar_solucion (const Operaciones_t &solucion)

  Muestra la solucion como las operaciones necesarias para llegar a la meta sin mostrar estados intermedios, es apta cuando los operadores son objetos funcion.

- template<typename Estado_t, typename Operaciones_t, typename NombreOperador_t> void ia::mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion, const NombreOperador_t &nombre_operadores)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son punteros a funciones.

- template<typename Estado_t, typename Operaciones_t> void ia::mostrar_estado_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solución como el estado alcanzado después de realizar las operaciones dadas al estado inicial dado, es apta tanto para operaciones como functores como con objetos función.

2.3.1. Documentación de las funciones

2.3.1.1. void ia::mostrar_estadisticas ()

Muestra las estadísticas de la última búsqueda realizada.

Si no se ha #definido IA_NO_ESTADISTICAS mostrará cero en todos los conteos. Si no se ha #definido IA_NO_CRONOMETRAR mostrará la cantidad de milisegundos transcurridos. Los milisegundos tienen una precisión de +/- 16 milisegundos, debido a que se usa la funcion GetTickCount de la API de windows y no se consulta directamente al procesador.

Ejemplos:
8puzzle.cpp, cantaros.cpp, laberinto.cpp, pastor.cpp, y reinas.cpp.

Definición en la línea 110 del archivo ia_ciega.h.

2.3.1.2. template<typename Estado_t, typename Operaciones_t, typename NombreOperador_t> void ia::mostrar_solucion (const Estado_t & *inicial*, const Operaciones_t & *solucion*, const NombreOperador_t & *nombre_operadores*)

Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son punteros a funciones.

Parámetros:

inicial Es el estado inicial que tenia el problema.

solucion Una secuencia (p.e. un vector) con las operaciones que nos llevan a la solucion

nombre_operadores Un mapa (p.e. NombresOperador) en que la llave es un puntero a la funcion y el valor, su nombre. Esta funcion muestra la cantidad de operaciones en solucion, el estado inicial y luego va aplicando las operaciones dadas en solucion al estado inicial para obtener los estados sucesivos y va mostrando el dato asociado a cada uno en el mapa nombre_operadores

Definición en la línea 172 del archivo ia_ciega.h.

2.3.1.3. template<typename Operaciones_t> void ia::mostrar_solucion (const Operaciones_t & *solucion*)

Muestra la solucion como las operaciones necesarias para llegar a la meta sin mostrar estados intermedios, es apta cuando los operadores son objetos funcion.

Parámetros:

solucion Una secuencia (p.e. un vector) con las operaciones que nos llevan a la solucion Cada miembro de la solucion debe tener un método get_descripción que devuelve el nombre del operador

Definición en la línea 152 del archivo ia_ciega.h.

2.3.1.4. template<typename Estado_t, typename Operaciones_t> void ia::mostrar_solucion (const Estado_t & *inicial*, const Operaciones_t & *solucion*)

Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son objetos función.

Parámetros:

inicial Es el estado inicial que tenia el problema.

solucion Una secuencia (p.e. un vector) con las operaciones que nos llevan a la solucion Cada miembro de la solucion debe tener un método get_descripción que devuelve el nombre del operador Esta función muestra la cantidad de operaciones en solucion, el estado inicial y luego va aplicando las operaciones dadas en solucion al estado inicial para obtener los estados sucesivos y los va mostrando

Ejemplos:

8puzzle.cpp, cantaros.cpp, laberinto.cpp, laberinto2.cpp, pastor.cpp, y viajero.cpp.

Definición en la línea 130 del archivo ia_ciega.h.

2.4. Búsquedas en grafos

Clases

- struct ia::Grafo< Nodo_t, Iterador_t, Costo_t >

  La estructura del grafo.

- struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >

  La estructura de los enlaces del grafo.

2.5. Detalles de implementación

Namespaces

- namespace ia::detalle

  Contiene detalles de implementación y la implementacion de los algoritmos de IA.

Clases

- class ia::SecuenciaEstados< Estado_t, Operaciones_t >::iterator

  Este iterador es el que da al contenedor SecuenciaEstados todas las características mencionadas en su descripción.

- struct ia::detalle::EstadoYRuta< Estado_t, Operaciones_t >

  Alias para contener varios estados cada uno con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t >

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t >

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_mejor_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::ComparaEstadosIgnorandoRutas< Estado_t, T >

  Functor auxiliar que compara un par <estado, ruta> tomando en cuenta solamente el estado.

- struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >::Auxiliar_costo_asignable_t

  Se utiliza para que el usuario pueda asignar el costo a un enlace de manera intuitiva.

Definiciones

- #define IA_IF_DEVUELVE_FALSO_REGISTRAR_SOLUCION(ruta, estado) if (! registrar_solucion || ! (*registrar_solucion)(ruta, estado))

  Para que podamos poner esta macro en vez del if largo, y que no tenga efecto si se ha deshabilitado el registro de soluciones usando IA_NO_REGISTRAR_SOLUCION.

Funciones

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::detalle::preferencia_amplitud (const Operaciones_t &operaciones, detalle::EstadoYRuta< Estado_t, Operaciones_t > &expansibles, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algorítmo de búsqueda por amplitud, no es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::preferencia_profundidad (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, bool &finalizar, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad, es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::profundidad_limitada (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t, typename Comparador_t> pair< Ruta_t, bool > ia::detalle::profundidad_limitada_doble (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &atajos, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &nuevos_atajos, bool(*registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada doble, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Comparador_mejor_t, typename Ruta_t, typename Estado_t, typename Operaciones_t> Ruta_t ia::detalle::primero_mejor (const Operaciones_t &operaciones, set< Estado_t > &visitados, detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t > &expansibles, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por primero el mejor (A), no es recursivo.*

2.5.1. Documentación de las funciones

2.5.1.1. template<typename Estado_t, typename Operaciones_t> vector<typename Operaciones_t::value_type> ia::detalle::preferencia_amplitud (const Operaciones_t & *operaciones*, detalle::EstadoYRuta< Estado_t, Operaciones_t > & *expansibles*, bool(*)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion*)

Algorítmo de búsqueda por amplitud, no es recursivo.

Parámetros:

operaciones Todas las operaciones disponibles

expansibles Lista de estados que hay que expandir inicialmente, normalmente el estado inicial y una ruta vacía

registrar_solucion Callback que se llamará cada vez que se halle una solución

Ejemplos:

cantaros.cpp, pastor.cpp, y viajero.cpp.

Definición en la línea 375 del archivo ia_ciega.h.

2.5.1.2. template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::preferencia_profundidad (const Operaciones_t & *operaciones*, set< Estado_t > & *visitados*, Ruta_t & *ruta*, const Estado_t & *actual*, unsigned int *profundidad*, bool & *finalizar*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion*)

Algoritmo de búsqueda por profundidad, es recursivo.

Además de los parámetros mencionados, si no se ha deshabilitado la obtención de estadísticas, existe un parámetro más llamado profundidad, utilizado para obtener la profundidad máxima que alcanza la búsqueda (esto se hace utilizando macros)

Parámetros:

operaciones Todas las operaciones disponibles

visitados Estados que ya se han visitado

ruta Ruta (operaciones realizadas) hasta el estado que se esta explorando

actual Estado que se esta explorando

finalizar Vale true ssi es necesario terminar el algoritmo (p.e. el callback indica que se debe detener la búsqueda)

registrar_solucion Callback llamado cada vez que se encuentra una solucion

Ejemplos:

cantaros.cpp.

Definición en la línea 433 del archivo ia_ciega.h.

2.5.1.3. template<typename Comparador_mejor_t, typename Ruta_t, typename Estado_t, typename Operaciones_t> Ruta_t ia::detalle::primero_mejor (const Operaciones_t & *operaciones*, set< Estado_t > & *visitados*, detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t > & *expansibles*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion*)

Algoritmo de búsqueda por primero el mejor (A∗), no es recursivo.

Parámetros:

operaciones Todas las operaciones disponibles

visitados Conjunto de estados visitados

expansibles Conjunto de estados que no se han explorado

registrar_solucion Callback llamado cada vez que se encuentra una solución

Definición en la línea 645 del archivo ia_ciega.h.

2.5.1.4. template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::profundidad_limitada (const Operaciones_t & *operaciones*, set< Estado_t > & *visitados*, Ruta_t & *ruta*, const Estado_t & *actual*, unsigned int *profundidad*, const unsigned int & *limite_profundidad*, bool & *finalizar*, bool(∗)(const vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion*)

Algoritmo de búsqueda por profundidad limitada, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

Parámetros:

operaciones Todas las operaciones disponibles

visitados Estados que ya se han visitado

ruta Ruta (operaciones realizadas) hasta el estado que se esta explorando

actual Estado que se esta explorando

profundidad Profundidad en la que se encuentra el estado actual

limite_profundidad Profundidad hasta la cual puede explorarse el árbol de estados

finalizar Vale true ssi es necesario terminar el algoritmo (p.e. el callback indica que se debe detener la búsqueda)

registrar_solucion Callback llamado cada vez que se encuentra una solucion

Ejemplos:

8puzzle.cpp, cantaros.cpp, pastor.cpp, reinas.cpp, y viajero.cpp.

Definición en la línea 500 del archivo ia_ciega.h.

Referenciado por ia::profundidad_iterativa().

2.5.1.5. template<typename Estado_t, typename Operaciones_t, typename Ruta_t, typename Comparador_t> pair<Ruta_t, bool> ia::detalle::profundidad_limitada_doble (const Operaciones_t & *operaciones*, set< Estado_t > & *visitados*, Ruta_t & *ruta*, const Estado_t & *actual*, unsigned int *profundidad*, const unsigned int & *limite_profundidad*, bool & *finalizar*, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > & *atajos*, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > & *nuevos_atajos*, bool(∗)(vector< typename Operaciones_t::value_type > &, const Estado_t &) *registrar_solucion*)

Algoritmo de búsqueda por profundidad limitada doble, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

Parámetros:

operaciones Todas las operaciones disponibles

visitados Conjunto de estados que ya han sido visitados

ruta Lista de operaciones realizadas para llegar al estado actual que debe explorarse

actual Estado actual que debe explorarse

profundidad Profundidad del nodo correspondiente al estado actual

limite_profundidad Máxima profundidad permisible

finalizar Vale true ssi la funcion debe terminar ya, p.e. porque el callback así lo indica

atajos Nodos que se hallan en el nivel más profundo en el arbol alterno (el que se recorre en el otro sentido)

nuevos_atajos Nodos que se hallan en el nivel más profundo de este árbol

registrar_solucion Callback llamado cada vez que se encuentra una solución

Ejemplos:

laberinto.cpp, y laberinto2.cpp.

Definición en la línea 574 del archivo ia_ciega.h.

Referenciado por ia::profundidad_limitada_doble().

3. IA Ciega 0.9 Documentación de namespace

3.1. Referencia del Namespace ia

3.1.1. Descripción detallada

Todo el contenido de esta libreria esta en este namespace.

Para usar las funciones, tipos, etc ponga la linea using namespace ia después de incluir las cabeceras, o use ia::funcion.

Clases

- struct Operacion

  Las operaciones pueden ser functores (objeto-funcion) que heredan de este, dándole como parámetro de plantilla el tipo del estado con el que opera.

- struct Operaciones

  La lista de operaciones se puede declarar como una instancia de esta estructura, que es un vector de punteros a las funciones que sirven de operadores.

- struct NombresOperadores

  Si se tiene un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, debería instanciar una estructura como esta, dando como parámetro de plantilla el tipo de estado con el que opera.

- struct OperadoresInversos

  Si utiliza las búsquedas bi-direccionales con un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, puede utilizar este mapa para establecer las operaciones inversas.

- class SecuenciaEstados

  Un contenedor que se construye en base al estado inicial y a una secuencia de operadores como las que devuelven las búsquedas; pero que al iterar devuelve los estados *generados al aplicar esas operaciones.*

- struct Grafo

  La estructura del grafo.

- struct Enlaces

  La estructura de los enlaces del grafo.

Namespaces

- namespace detalle

  Contiene detalles de implementación y la implementacion de los algoritmos de IA.

Funciones

- void mostrar_estadisticas ()

  Muestra las estadísticas de la última búsqueda realizada.

- template<typename Estado_t, typename Operaciones_t> void mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son objetos función.

- template<typename Operaciones_t> void mostrar_solucion (const Operaciones_t &solucion)

  Muestra la solucion como las operaciones necesarias para llegar a la meta sin mostrar estados intermedios, es apta cuando los operadores son objetos funcion.

- template<typename Estado_t, typename Operaciones_t, typename NombreOperador_t> void mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion, const NombreOperador_t &nombre_operadores)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son punteros a funciones.

- template<typename Estado_t, typename Operaciones_t> void mostrar_estado_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solución como el estado alcanzado después de realizar las operaciones dadas al estado inicial dado, es apta tanto para operaciones como functores como con objetos función.

- template<typename Operaciones_t> vector< typename Operaciones_t::value_type > simplificar_bidireccional (const pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > &solucion, const OperadoresInversos< Operaciones_t > &inversos)

  Recibe una solución bidireccional y devuelve una solucion secuencial.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > preferencia_amplitud (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de preferencia por amplitud.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > preferencia_profundidad (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Función para buscar una o más soluciones usando el método de preferencia por profundidad (depth-first-search).

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > profundidad_limitada (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad limitada.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > profundidad_iterativa (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad iterativa.

- template<typename Comparador_mejor_t, typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > primero_mejor (const Operaciones_t &operaciones, const Estado_t &inicial, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando el algoritmo primero mejor.

- template<typename Estado_t, typename Operaciones_t> pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > profundidad_limitada_doble (const Operaciones_t &operaciones, const Estado_t &inicial, const Estado_t &final, const unsigned int &limite_profundidad, bool(∗registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando un algoritmo similar al de profundidad limitada, pero empezando desde el estado inicial y el estado final.

- template<typename Nodo_t, typename Iterador_t, typename Costo_t> vector< Iterador_t > vecino_mas_proximo (Grafo< Nodo_t, Iterador_t, Costo_t > &grafo, typename Grafo< Nodo_t, Iterador_t, Costo_t >::iterator &it_nodo)
- template<typename Nodo_t, typename Iterador_t, typename Costo_t> vector< Iterador_t > vecino_mas_proximo (Grafo< Nodo_t, Iterador_t, Costo_t > &grafo, const Nodo_t &nodo)

3.1.2. Documentación de las funciones

3.1.2.1. template<typename Operaciones_t> vector<typename Operaciones_t::value_type> ia::simplificar_bidireccional (const pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > & *solucion*, const OperadoresInversos< Operaciones_t > & *inversos*)

Recibe una solución bidireccional y devuelve una solucion secuencial.

Parámetros:

solucion Solucion devuelta por un algorítmo de búsqueda bidireccional

inversos Mapa con que asocia a cada operación-adelante una operación-atras

Devuelve:

Devuelve una solucion como la que se hubiera hallado utilizando un algorítmo no bidireccional

Ejemplos:

laberinto.cpp, y laberinto2.cpp.

Definición en la línea 349 del archivo ia_ciega.h.

3.2. Referencia del Namespace ia::detalle

3.2.1. Descripción detallada

Contiene detalles de implementación y la implementacion de los algoritmos de IA.

Para usar los algoritmos de búsqueda use una de las funciones de interfaz que están en el namespace ia

Ver también:

ia

Clases

- struct EstadoYRuta

  Alias para contener varios estados cada uno con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct SetEstadoYRuta

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct MultisetEstadoYRuta

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_mejor_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ComparaEstadosIgnorandoRutas

  Functor auxiliar que compara un par <estado, ruta> tomando en cuenta solamente el estado.

Funciones

- void reiniciar_estadisticas ()
- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > preferencia_amplitud (const Operaciones_t &operaciones, detalle::EstadoYRuta< Estado_t, Operaciones_t > &expansibles, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algorítmo de búsqueda por amplitud, no es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t preferencia_profundidad (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, bool &finalizar, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad, es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t profundidad_limitada (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t, typename Comparador_t> pair< Ruta_t, bool > profundidad_limitada_doble (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &atajos, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &nuevos_atajos, bool(*registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada doble, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Comparador_mejor_t, typename Ruta_t, typename Estado_t, typename Operaciones_t> Ruta_t primero_mejor (const Operaciones_t &operaciones, set< Estado_t > &visitados, detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t > &expansibles, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por primero el mejor (A), no es recursivo.*

4. IA Ciega 0.9 Documentación de clases

4.1. Referencia de la Estructura ia::detalle::ComparaEstadosIgnorandoRutas< Estado_t, T >

`#include <ia_ciega.h>`

4.1.1. Descripción detallada

template<typename Estado_t, typename T> struct ia::detalle::ComparaEstadosIgnorandoRutas< Estado_t, T >

Functor auxiliar que compara un par <estado, ruta> tomando en cuenta solamente el estado.

Se usa en la búsqueda bidireccional

Definición en la línea 691 del archivo ia_ciega.h.

Métodos públicos

- bool operator() (const pair< Estado_t, T > &izq, const pair< Estado_t, T > &der) const

4.2. Referencia de la Estructura ia::Enlaces< Nodo_t, Iterador_t, Costo_t >

`#include <ia_ciega.h>`

Herencias std::multimap< Costo_t, const Iterador_t * >.

4.2.1. Descripción detallada

template<typename Nodo_t, typename Iterador_t, typename Costo_t> struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >

La estructura de los enlaces del grafo.

Definición en la línea 994 del archivo ia_ciega.h.

Tipos públicos

- typedef Grafo< Nodo_t, Iterador_t, Costo_t > Grafo_t
- typedef multimap< Costo_t, const Iterador_t * > Padre_t
- typedef multimap< Costo_t, const Iterador_t * >::iterator iterator

Métodos públicos

- Enlaces (Grafo< Nodo_t, Iterador_t, Costo_t > &g)
- Padre_t::iterator begin ()
- Padre_t::iterator end ()
- Auxiliar_costo_asignable_t operator[ ] (Nodo_t const &destino)

Atributos públicos

- Grafo_t & grafo

Amigas

- struct Auxiliar_costo_asignable_t

Clases

- struct Auxiliar_costo_asignable_t

  Se utiliza para que el usuario pueda asignar el costo a un enlace de manera intuitiva.

4.3. Referencia de la Estructura ia::Enlaces< Nodo_t, Iterador_t, Costo_t >::Auxiliar_costo_asignable_t

`#include <ia_ciega.h>`

4.3.1. Descripción detallada

template<typename Nodo_t, typename Iterador_t, typename Costo_t> struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >::Auxiliar_costo_asignable_t

Se utiliza para que el usuario pueda asignar el costo a un enlace de manera intuitiva.

Definición en la línea 1005 del archivo ia_ciega.h.

Métodos públicos

- Auxiliar_costo_asignable_t (Enlaces &enlaces, typename Enlaces::iterator par_costo_)
- void operator= (const Costo_t &costo)
- operator Costo_t ()

Atributos públicos

- Padre_t::iterator par_costo
- Padre_t & outer

4.4. Referencia de la Estructura ia::detalle::EstadoYRuta< Estado_t, Operaciones_t >

`#include <ia_ciega.h>`

Herencias std::vector< pair< Estado_t, vector< Operaciones_t::value_type > > >.

4.4.1. Descripción detallada

template<typename Estado_t, typename Operaciones_t> struct ia::detalle::EstadoYRuta< Estado_t, Operaciones_t >

Alias para contener varios estados cada uno con las operaciones realizadas para llegar a él (o sea, su ruta).

Definición en la línea 366 del archivo ia_ciega.h.

4.5. Referencia de la Estructura ia::Grafo< Nodo_t, Iterador_t, Costo_t >

`#include <ia_ciega.h>`

Herencias std::map< Nodo_t, Enlaces< Nodo_t, Iterador_t, Costo_t > >.

4.5.1. Descripción detallada

template<typename Nodo_t, typename Iterador_t, typename Costo_t = int> struct ia::Grafo< Nodo_t, Iterador_t, Costo_t >

La estructura del grafo.

Internamente se almacena como un mapa de nodos, cada uno asociado a sus enlaces, de tipo Enlaces. Los parametros de plantilla son estos:

Parámetros:

Nodo_t Es el tipo del nodo o vértice con el que el grafo debe funcionar. Es un tipo creado por el usuario de la librería para su propio dominio de problema

Iterador_t Es el tipo de iterador utilizado por el grafo. Debe ser especificado por el usuario como se indica más abajo

Costo_t Es el tipo del costo de cada enlace, por defecto es un entero (int)

Definición en la línea 976 del archivo ia_ciega.h.

Tipos públicos

- typedef Enlaces< Nodo_t, Iterador_t, Costo_t > Enlaces_t

  Un valor de este tipo esta asociado a cada nodo y permite acceder a sus vecinos y conocer los valores asociados al enlace (costo).

Métodos públicos

- Enlaces_t & operator[ ] (const Nodo_t &origen)

  El operator[ ] permite acceder a los enlaces de un nodo de manera intuitiva.

- ~Grafo ()

  El destructor libera recursos utilizados internamente: elimina los punteros a los iteradores que se crearon en los enlaces.

4.6. Referencia de la Estructura ia::detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t >

`#include <ia_ciega.h>`

Herencias std::multiset< pair< Estado_t, Ruta_t >, Comparador_mejor_t >.

4.6.1. Descripción detallada

template<typename Estado_t, typename Ruta_t, typename Comparador_mejor_t> struct ia::detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t >

Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_mejor_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

Ya que es un multiset es necesario que comparador_t sea un functor que sea un strict-weak-ordering (como se define para la STL), sin embargo no es necesario que si no(a < b) y no(b < a) entonces a y b representen al mismo estado, por otra parte el resultado del functor debe ser determinístico

Definición en la línea 635 del archivo ia_ciega.h.

4.7. Referencia de la Estructura ia::NombresOperadores< Estado_t >

`#include <ia_ciega.h>`

Herencias std::map< bool(∗)(const Estado_t &, Estado_t &), string >.

4.7.1. Descripción detallada

template<typename Estado_t> struct ia::NombresOperadores< Estado_t >

Si se tiene un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, debería instanciar una estructura como esta, dando como parámetro de plantilla el tipo de estado con el que opera.

La llave del mapa es un puntero a una función, y su valor asociado es el nombre o descripción del operador

Ejemplos:
cantaros.cpp, laberinto.cpp, laberinto2.cpp, y pastor.cpp.

Definición en la línea 235 del archivo ia_ciega.h.

4.8. Referencia de la Estructura ia::Operacion< Estado_t >

`#include <ia_ciega.h>`

4.8.1. Descripción detallada

template<typename Estado_t> struct ia::Operacion< Estado_t >

Las operaciones pueden ser functores (objeto-funcion) que heredan de este, dándole como parámetro de plantilla el tipo del estado con el que opera.

La lista de operaciones puede ser un vector de punteros a este tipo. Nota: El uso de esta estructura como clase base implicará una llamada a un método virtual al aplicar cada operación, no se recomienda su uso si el tiempo es importante; considere crear las operaciones como distintas instancias de *una misma clase*.

Ejemplos:
8puzzle.cpp, reinas.cpp, y viajero.cpp.

Definición en la línea 213 del archivo ia_ciega.h.

Métodos públicos

- Operacion (const string &nombre_=string())
- virtual bool operator() (const Estado_t &antes, Estado_t &despues) const =0

  Debe aplicar la operacion sobre el estado 'antes' y poner el estado resultante en 'despues'.

- virtual string & get_descripcion ()

Atributos públicos

- string nombre

4.8.2. Documentación de las funciones miembro

4.8.2.1. template<typename Estado_t> virtual bool ia::Operacion< Estado_t >::operator() (const Estado_t & *antes*, Estado_t & *despues*) const [pure virtual]

Debe aplicar la operacion sobre el estado 'antes' y poner el estado resultante en 'despues'.

Devuelve:
true ssi la operación se llevó a cabo con éxito; si devuelve false el valor que tenga el parametro 'despues' es irrelevante

Ejemplos:
8puzzle.cpp, reinas.cpp, y viajero.cpp.

4.9. Referencia de la Estructura ia::Operaciones< Estado_t >

`#include <ia_ciega.h>`

Herencias std::vector< bool(*)(const Estado_t &, Estado_t &) >.

4.9.1. Descripción detallada

template<typename Estado_t> struct ia::Operaciones< Estado_t >

La lista de operaciones se puede declarar como una instancia de esta estructura, que es un vector de punteros a las funciones que sirven de operadores.

Ejemplos:
cantaros.cpp, laberinto.cpp, laberinto2.cpp, pastor.cpp, y reinas.cpp.

Definición en la línea 228 del archivo ia_ciega.h.

4.10. Referencia de la Estructura ia::OperadoresInversos< Operaciones_t >

`#include <ia_ciega.h>`

Herencias std::map< Operaciones_t::value_type, Operaciones_t::value_type >.

4.10.1. Descripción detallada

template<typename Operaciones_t> struct ia::OperadoresInversos< Operaciones_t >

Si utiliza las búsquedas bi-direccionales con un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, puede utilizar este mapa para establecer las operaciones inversas.

Ejemplos:
laberinto.cpp, y laberinto2.cpp.

Definición en la línea 241 del archivo ia_ciega.h.

Métodos públicos

- OperadoresInversos< Operaciones_t > & complementar ()

  Si solo construye los inversos de la mitad de los operadores, use este método para complementar con los operadores restantes ej: siendo a,b,c operadores y A, B, C sus operadores inversos, entonces use una instancia inv de esta estructura y ponga inv[a] = A; inv[b] = B; inv[c] = C; luego llamar a inv.complementar() añadirá inv[A] = a; inv[B] = b; inv[C] = c.

4.10.2. Documentación de las funciones miembro

4.10.2.1. template<typename Operaciones_t> OperadoresInversos<Operaciones_t>& ia::OperadoresInversos< Operaciones_t >::complementar () [inline]

Si solo construye los inversos de la mitad de los operadores, use este método para complementar con los operadores restantes ej: siendo a,b,c operadores y A, B, C sus operadores inversos, entonces use una instancia inv de esta estructura y ponga inv[a] = A; inv[b] = B; inv[c] = C; luego llamar a inv.complementar() añadirá inv[A] = a; inv[B] = b; inv[C] = c.

Como este método devuelve una referencia a la misma clase se puede utilizar como rvalue, por ejemplo: mostrar_solucion(...., inv.complementar())

Definición en la línea 248 del archivo ia_ciega.h.

4.11. Referencia de la Clase ia::SecuenciaEstados< Estado_t, Operaciones_t >

`#include <ia_ciega.h>`

4.11.1. Descripción detallada

template<typename Estado_t, typename Operaciones_t> class ia::SecuenciaEstados< Estado_t, Operaciones_t >

Un contenedor que se construye en base al estado inicial y a una secuencia de operadores como las que devuelven las búsquedas; pero que al iterar devuelve los *estados* generados al aplicar esas operaciones.

Para las búsquedas bidimensionales use simplificar_bidireccional al resultado para obtener una lista como la requerida por esta clase. Es un contenedor especial, similar a un Forward Container pero que no es propietario de sus elementos, dereferenciar a su iterador en realidad aplica una operación al estado actual y devuelve el nuevo estado obtenido, modificando también al estado actual. Vea los comentarios de la clase interna (inner class) iterator en el código fuente para mayores detalles .

Ver también:
simplificar_bidireccional

Ejemplos:
laberinto2.cpp.

Definición en la línea 267 del archivo ia_ciega.h.

Métodos públicos

- SecuenciaEstados (const Estado_t &inicial_, const vector< typename Operaciones_t::value_type > &operaciones_)
- const iterator begin ()
- const iterator end ()
- unsigned int size ()

Amigas

- class iterator

Clases

- class iterator

  Este iterador es el que da al contenedor SecuenciaEstados todas las características mencionadas en su descripción.

4.12. Referencia de la Clase ia::SecuenciaEstados< Estado_t, Operaciones_t >::iterator

`#include <ia_ciega.h>`

Herencias iterator.

4.12.1. Descripción detallada

template<typename Estado_t, typename Operaciones_t> class ia::SecuenciaEstados< Estado_t, Operaciones_t >::iterator

Este iterador es el que da al contenedor SecuenciaEstados todas las características mencionadas en su descripción.

Definición en la línea 279 del archivo ia_ciega.h.

Métodos públicos

- iterator (Estado_t &inicial_, const Ruta_t &operaciones_)
- iterator (const Ruta_t &operaciones_)
- bool operator== (const iterator &x) const
- bool operator!= (const iterator &x) const
- Estado_t operator * () const
- iterator & operator++ ()
- iterator operator++ (int)
- iterator & operator– ()
- iterator operator– (int)

4.13. Referencia de la Estructura ia::detalle::SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t >

`#include <ia_ciega.h>`

Herencias std::set< pair< Estado_t, Ruta_t >, Comparador_t >.

4.13.1. Descripción detallada

template<typename Estado_t, typename Ruta_t, typename Comparador_t> struct ia::detalle::SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t >

Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

Como es un set es necesario que comparador_t sea un functor que sea un strict-weak-ordering (como se define para la STL), es importante que diferencie correctamente un estado de otro (o sea que si no(a < b) y no(b < a) entonces a y b representan al mismo estado)

Definición en la línea 558 del archivo ia_ciega.h.

5. IA Ciega 0.9 Documentación de archivos

5.1. Referencia del Archivo include/ia_ciega.h

5.1.1. Descripción detallada

Autor:
Roberto Oropeza Gamarra Contiene a toda la libreria IA Ciega Mejora/Correccion/Optimizacion: Xxxxx añadio/modifico/quito yyyyy para/por zzzzz

Definición en el archivo ia_ciega.h.

```
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <map>
#include <ctime>
#include <cassert>
#include <windows.h>
```

Namespaces

- namespace ia
- namespace ia::detalle

Clases

- struct ia::Operacion< Estado_t >

  Las operaciones pueden ser functores (objeto-funcion) que heredan de este, dándole como parámetro de plantilla el tipo del estado con el que opera.

- struct ia::Operaciones< Estado_t >

  La lista de operaciones se puede declarar como una instancia de esta estructura, que es un vector de punteros a las funciones que sirven de operadores.

- struct ia::NombresOperadores< Estado_t >

Si se tiene un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, debería instanciar una estructura como esta, dando como parámetro de plantilla el tipo de estado con el que opera.

- struct ia::OperadoresInversos< Operaciones_t >

  Si utiliza las búsquedas bi-direccionales con un vector con punteros a funciones para representar las operaciones que se pueden aplicar al realizar las búsquedas, puede utilizar este mapa para establecer las operaciones inversas.

- class ia::SecuenciaEstados< Estado_t, Operaciones_t >

  Un contenedor que se construye en base al estado inicial y a una secuencia de operadores como las que devuelven las búsquedas; pero que al iterar devuelve los estados *generados al aplicar esas operaciones.*

- class ia::SecuenciaEstados< Estado_t, Operaciones_t >::iterator

  Este iterador es el que da al contenedor SecuenciaEstados todas las características mencionadas en su descripción.

- struct ia::detalle::EstadoYRuta< Estado_t, Operaciones_t >

  Alias para contener varios estados cada uno con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t >

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t >

  Alias para tener varios estados ordenados de acuerdo a cierto criterio (dado por Comparador_mejor_t), cada uno asociado con las operaciones realizadas para llegar a él (o sea, su ruta).

- struct ia::detalle::ComparaEstadosIgnorandoRutas< Estado_t, T >

  Functor auxiliar que compara un par <estado, ruta> tomando en cuenta solamente el estado.

- struct ia::Grafo< Nodo_t, Iterador_t, Costo_t >

  La estructura del grafo.

- struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >

  La estructura de los enlaces del grafo.

- struct ia::Enlaces< Nodo_t, Iterador_t, Costo_t >::Auxiliar_costo_asignable_t

  Se utiliza para que el usuario pueda asignar el costo a un enlace de manera intuitiva.

Definiciones

- #define IA_IF_DEVUELVE_FALSO_REGISTRAR_SOLUCION(ruta, estado) if (! registrar_solucion || ! (*registrar_solucion)(ruta, estado))

  Para que podamos poner esta macro en vez del if largo, y que no tenga efecto si se ha deshabilitado el registro de soluciones usando IA_NO_REGISTRAR_SOLUCION.

- #define IA_INICIO_CRONOMETRO ia::detalle::cronometro = GetTickCount();
- #define IA_FIN_CRONOMETRO ia::detalle::cronometro = GetTickCount() - ia::detalle::cronometro;
- #define IA_INCREMENTAR_SOLUCIONES detalle::conteo_soluciones ++;
- #define IA_INCREMENTAR_EXPANDIDOS detalle::conteo_expandidos ++;
- #define IA_INCREMENTAR_VISITADOS detalle::conteo_visitados ++;
- #define IA_INCREMENTAR_PROFUNDIDAD_MAXIMA detalle::profundidad_maxima ++;
- #define IA_ACTUALIZAR_PROFUNDIDAD_MAXIMA(X) if (detalle::profundidad_maxima < (X)) detalle::profundidad_maxima = X;
- #define IA_REINICIAR_ESTADISTICAS detalle::reiniciar_estadisticas();
- #define IA_REINICIAR_VISITADOS detalle::conteo_visitados = 0;

Funciones

- void ia::detalle::reiniciar_estadisticas ()
- void ia::mostrar_estadisticas ()

  Muestra las estadísticas de la última búsqueda realizada.

- template<typename Estado_t, typename Operaciones_t> void ia::mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son objetos función.

- template<typename Operaciones_t> void ia::mostrar_solucion (const Operaciones_t &solucion)

  Muestra la solucion como las operaciones necesarias para llegar a la meta sin mostrar estados intermedios, es apta cuando los operadores son objetos funcion.

- template<typename Estado_t, typename Operaciones_t, typename NombreOperador_t> void ia::mostrar_solucion (const Estado_t &inicial, const Operaciones_t &solucion, const NombreOperador_t &nombre_operadores)

  Muestra la solucion como los estados intermedios entre el estado inicial y la meta así como la operación aplicada a cada uno para obtener el siguiente, es apta cuando los operadores son punteros a funciones.

- template<typename Estado_t, typename Operaciones_t> void ia::mostrar_estado_solucion (const Estado_t &inicial, const Operaciones_t &solucion)

  Muestra la solución como el estado alcanzado después de realizar las operaciones dadas al estado inicial dado, es apta tanto para operaciones como functores como con objetos función.

- template<typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::simplificar_bidireccional (const pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > &solucion, const OperadoresInversos< Operaciones_t > &inversos)

  Recibe una solución bidireccional y devuelve una solucion secuencial.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::detalle::preferencia_amplitud (const Operaciones_t &operaciones, detalle::EstadoYRuta< Estado_t, Operaciones_t > &expansibles, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algorítmo de búsqueda por amplitud, no es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::preferencia_profundidad (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, bool &finalizar, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad, es recursivo.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t> Ruta_t ia::detalle::profundidad_limitada (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, bool(∗registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Estado_t, typename Operaciones_t, typename Ruta_t, typename Comparador_t> pair< Ruta_t, bool > ia::detalle::profundidad_limitada_doble (const Operaciones_t &operaciones, set< Estado_t > &visitados, Ruta_t &ruta, const Estado_t &actual, unsigned int profundidad, const unsigned int &limite_profundidad, bool &finalizar, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &atajos, SetEstadoYRuta< Estado_t, Ruta_t, Comparador_t > &nuevos_atajos, bool(∗registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por profundidad limitada doble, es recursivo pero solo tiene una variable Estado_t en el stack, el resto son estáticas.

- template<typename Comparador_mejor_t, typename Ruta_t, typename Estado_t, typename Operaciones_t> Ruta_t ia::detalle::primero_mejor (const Operaciones_t &operaciones, set< Estado_t > &visitados, detalle::MultisetEstadoYRuta< Estado_t, Ruta_t, Comparador_mejor_t > &expansibles, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &))

  Algoritmo de búsqueda por primero el mejor (A), no es recursivo.*

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::preferencia_amplitud (const Operaciones_t &operaciones, const Estado_t &inicial, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de preferencia por amplitud.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::preferencia_profundidad (const Operaciones_t &operaciones, const Estado_t &inicial, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Función para buscar una o más soluciones usando el método de preferencia por profundidad (depth-first-search).

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::profundidad_limitada (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad limitada.

- template<typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::profundidad_iterativa (const Operaciones_t &operaciones, const Estado_t &inicial, const unsigned int &limite_profundidad, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Funcion para buscar una o mas soluciones usando el método de profundidad iterativa.

- template<typename Comparador_mejor_t, typename Estado_t, typename Operaciones_t> vector< typename Operaciones_t::value_type > ia::primero_mejor (const Operaciones_t &operaciones, const Estado_t &inicial, bool(*registrar_solucion)(const vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando el algoritmo primero mejor.

- template<typename Estado_t, typename Operaciones_t> pair< vector< typename Operaciones_t::value_type >, vector< typename Operaciones_t::value_type > > ia::profundidad_limitada_doble (const Operaciones_t &operaciones, const Estado_t &inicial, const Estado_t &final, const unsigned int &limite_profundidad, bool(*registrar_solucion)(vector< typename Operaciones_t::value_type > &, const Estado_t &)=NULL)

  Busca en el espacio de estados usando un algoritmo similar al de profundidad limitada, pero empezando desde el estado inicial y el estado final.

- template<typename Nodo_t, typename Iterador_t, typename Costo_t> vector< Iterador_t > ia::vecino_mas_proximo (Grafo< Nodo_t, Iterador_t, Costo_t > &grafo, typename Grafo< Nodo_t, Iterador_t, Costo_t >::iterator &it_nodo)
- template<typename Nodo_t, typename Iterador_t, typename Costo_t> vector< Iterador_t > ia::vecino_mas_proximo (Grafo< Nodo_t, Iterador_t, Costo_t > &grafo, const Nodo_t &nodo)

6. IA Ciega 0.9 Documentación de ejemplos

6.1. 8puzzle.cpp

Usa functores (objetos-función) para especificar las operaciones sobre los estados. Cada movimiento (operacion) es una instancia de la clase Mover.

Ademas de usar `vector` y `map` y `set` de la STL (`set` es un conjunto ordenado en que no pueden repetirse los elementos) usa los algoritmos `lexicographical_compare` que compara dos contenedores a modo de diccionario, es decir que si ambos son identicos solo que uno de los contenedores tiene menos elementos, este está primero. Tambien se usan `copy` y `back_inserter`

que juntos añaden el contenido del primer contenedor al final del segundo. `random_shuffle` desordena de manera aleatoria todo el contenido de un contenedor.

Utiliza la búsqueda de profundidad limitada

```
1 /* Resuelve el 8-puzzle
2 */
3
4 #include <algorithm>
5 #include <iostream>
6 #include <iomanip>
7 #include <cassert>
8
9 //#define IA_NO_CRONOMETRAR
10 #include "../include/ia_ciega.h"
11
12 using namespace std;
13 using namespace ia;
14
15 const int ANCHO = 3;
16 const int ALTO = 3;
17
18 // matrizFicha[2][1] = 6 quiere decir que en la fila 3 columna 2 está la ficha #6
19 typedef int MatrizFicha[ALTO][ANCHO];
20
21 // Este es el estado
22 struct Puzzle {
23    MatrizFicha ficha;
24    Puzzle() {}
25    Puzzle(bool tonta) { // El parametro tonto hace que se pongan las fichas en el puzzle
26       int numero = 0;
27       for ( int fila = 0; fila < ALTO; ++ fila )
28          for ( int col = 0; col < ANCHO; ++ col )
29             ficha[fila][col] = ++numero;
30       ficha[ALTO-1][ANCHO-1] = 0; // La ultima queda vacía
31    }
32    bool operator<(const Puzzle& otro) const {
33       return lexicographical_compare( ficha[0], ficha[0] + ALTO * ANCHO, otro.ficha[0], otro.ficha[0] + ALTO * ANCHO
34    }
35    bool es_meta() const {
36       int numero = 0;
37       for ( int fila = 0; fila < ALTO; ++ fila )
38          for ( int col = 0; col < ANCHO; ++ col )
39             if ( ficha[fila][col] != ++numero && numero != ANCHO*ALTO)
40                return false;
41       return true;
42    }
43    friend ostream& operator<<(ostream& os, const Puzzle& t ) {
44       os << "\n";
45       for( int fila = 0; fila < ALTO; ++ fila ) {
46          os << "[";
47          for( int col = 0; col < ANCHO; ++ col )
48             os << setw(4) << (unsigned int)t.ficha[fila][col];
49          os << "]\n";
50       }
51       return os;
52    }
53 };
54
55 struct Mover : public Operacion<Puzzle> {
56    enum Direccion {ARRIBA, DERECHA, ABAJO, IZQUIERDA};
57    Direccion dir;
58    unsigned int fila_orig, col_orig; // posicion desde la que se mueve la ficha
59    int fila_des, col_des; // posición a la que llega
60    // Para inicializar se dice la posición inicial de la ficha y la dirección del movimento
61    Mover( const unsigned int& fila_, const unsigned int& col_, const Direccion& dir_ ) : fila_orig( fila_ ), col_ori
62       switch ( dir ) {
63          case ARRIBA:
64             assert( fila_orig != 0 );
65             fila_des = fila_orig - 1;            col_des = col_orig;
66             break;
```

```
67          case DERECHA:
68             assert( col_orig != ANCHO - 1 );
69             fila_des = fila_orig;               col_des = col_orig + 1;
70             break;
71          case ABAJO:
72             assert( fila_orig != ALTO - 1 );
73             fila_des = fila_orig + 1;          col_des = col_orig;
74             break;
75          case IZQUIERDA:
76             assert( col_orig != 0 );
77             fila_des = fila_orig;              col_des = col_orig - 1;
78             break;
79       }
80    }
81    bool operator()( const Puzzle& antes, Puzzle& despues ) const {
82       if ( antes.ficha[fila_orig][col_orig] == 0 || antes.ficha[fila_des][col_des] != 0 )
83          return false;
84       despues = antes;
85       despues.ficha[fila_des][col_des] = antes.ficha[fila_orig][col_orig];
86       despues.ficha[fila_orig][col_orig] = 0;
87       return true;
88    }
89 };
90
91
92 int main() {
93    Puzzle a, b(true);
94    typedef vector<Mover*> Movimientos;
95    Movimientos operaciones;
96
97    // Generar las operaciones
98    for( int fila = 0; fila < ALTO; ++ fila )
99       for( int col = 0; col < ANCHO; ++ col ) {
100          if ( fila != 0 )
101             operaciones.push_back( new Mover( fila, col, Mover::ARRIBA ) );
102          if ( col != ANCHO - 1 )
103             operaciones.push_back( new Mover( fila, col, Mover::DERECHA ) );
104          if ( fila != ALTO - 1 )
105             operaciones.push_back( new Mover( fila, col, Mover::ABAJO ) );
106          if ( col != 0 )
107             operaciones.push_back( new Mover( fila, col, Mover::IZQUIERDA ) );
108       }
109
110    Puzzle ordenado(true), inicial;
111
112    // Mezclar el puzzle
113    cout << "Mezclando...\n";
114    Movimientos mezcolanza( operaciones );
115    for ( unsigned long i = 0; i < 100; ++ i )
116       copy( operaciones.begin(), operaciones.end(), back_inserter( mezcolanza ) );
117    set<Puzzle> intermedios; // se usa para que no se retroceda al mezclar
118    intermedios.insert( ordenado );
119    for( int paso = 0; paso < 10; ++ paso ) {
120       random_shuffle(mezcolanza.begin(), mezcolanza.end() );
121       for ( int i = 0; i < mezcolanza.size(); ++ i )
122          if ( (*mezcolanza[i])(ordenado, inicial) && intermedios.find(inicial) == intermedios.end() ) {
123             intermedios.insert( inicial );
124             ordenado = inicial;
125          }
126    }
127    cout << "Se mezclo con " << intermedios.size() << " movidas\n";
128    intermedios.clear();
129    cout << inicial;
130
131    mostrar_solucion( inicial, profundidad_limitada(operaciones, inicial,30) );
132    mostrar_estadisticas();
133 }
```

6.2. cantaros.cpp

Usa punteros a funciones para especificar las operaciones realizables sobre los estados. Demuestra cómo poner nombres a las operaciones que estan dadas como punteros a funciones.

Lo único que usa de la STL es el tipo `vector` y el `map`, sobre el vector solo usa el metodo `push_back` que sirve para aumentar un valor más al vector, y del `map` usa el operador corchetes (`operator[]`), que hace que un `map` se pueda usar más o menos como un vector.

Utiliza las búsquedas de preferencia en profundidad, profundidad limitada, preferencia por amplitud y profundidad iterativa.

Resuelve el problema de los cántaros, en el que se tienen dos cántaros, no aforados (sin marcas de medición), El cántaro A tiene 3 litros de capacidad, y B 4 litros. Tambien hay una pila abierta, y vale mojar el piso. Al principio ambos cántaros estan vacíos. Hay que llenar el cantaro B con exactamente 2 litros.

```
1 /* Resuelve el problema de los cántaros:
2     Se tienen dos cántaros, no aforados (sin marcas de medición),
3     El cántaro A tiene 3 litros de capacidad, y B 4 litros. Tambien hay una pila abierta, y vale mojar el piso.
4     Al principio ambos cántaros estan vacíos.
5     Hay que llenar el cantaro B con exactamente 2 litros.
6 */
7
8 #include <string>
9 #include <vector>
10 #include <iostream>
11
12 #define IA_NO_CRONOMETRAR
13 #include "../include/ia_ciega.h"
14
15 using namespace std;
16 using namespace ia;
17
18 // Capacidad de cada cántaro
19 const unsigned int MAX_A = 3;
20 const unsigned int MAX_B = 4;
21
22 struct Estado {
23    // indica la cantidad de agua en el cantaro A y B
24    unsigned int a, b;
25    Estado() {}
26    Estado( int a_, int b_ ) : a(a_), b(b_) {}
27    bool es_meta() const {
28       return b == 2;
29    }
30    bool operator<( const Estado& otro ) const {
31       return (a * 10 + b) < (otro.a * 10 + otro.b);
32    }
33    friend ostream& operator<<(ostream& os, Estado& e ) {
34       os << "A con " << e.a << " litros; B con " << e.b << " litros.";
35       return os;
36    }
37 };
38
39 bool vaciarAaB( const Estado& antes, Estado& despues ) {
40       if ( ! antes.a || antes.a + antes.b > MAX_B ) // Si A esta vacío o rebalsaría B
41       return false;
42    despues.b = antes.b + antes.a;
43    despues.a = 0;
44    return true;
45 }
46
47 bool vaciarBaA(const Estado& antes, Estado& despues ) {
48    if ( ! antes.b || antes.a + antes.b > MAX_A ) // Si B esta vacío o rebalsaría A
49       return false;
50    despues.a = antes.b + antes.a;
51    despues.b = 0;
52    return true;
53 }
54
55 bool llenarA(const Estado& antes, Estado& despues ) {
```

```
56    if ( antes.a == MAX_A )
57       return false;
58    despues.a = MAX_A;
59    despues.b = antes.b;
60    return true;
61 }
62
63 bool llenarB(const Estado& antes, Estado& despues ) {
64    if ( antes.b == MAX_B )
65       return false;
66    despues.a = antes.a;
67    despues.b = MAX_B;
68    return true;
69 }
70
71
72 // Verter de A a B hasta llenar B
73 bool verterAllenandoB(const Estado& antes, Estado& despues ) {
74    if ( antes.b + antes.a < MAX_B )
75       return false;
76    despues.a = antes.a - ( MAX_B - antes.b );
77    despues.b = MAX_B;
78    return true;
79 }
80
81 // Verter de B a A hasta llenar A
82 bool verterBllenandoA(const Estado& antes, Estado& despues ) {
83    if ( antes.a + antes.b < MAX_A )
84       return false;
85    despues.a = MAX_A;
86    despues.b = antes.b - ( MAX_A - antes.a);
87    return true;
88 }
89
90 // Vaciar A al piso
91 bool vaciarA(const Estado& antes, Estado& despues ) {
92    if ( ! antes.a )
93       return false;
94    despues.a = 0;
95    despues.b = antes.b;
96    return true;
97 }
98
99 // Vaciar B al piso
100 bool vaciarB(const Estado& antes, Estado& despues ) {
101    if ( ! antes.b )
102       return false;
103    despues.a = antes.a;
104    despues.b = 0;
105    return true;
106 }
107
108
109 int main() {
110    Operaciones<Estado> operaciones;
111    NombresOperadores<Estado> nombres;
112    operaciones.push_back( &vaciarAaB ); nombres[vaciarAaB] = "Vaciar de A a B";
113    operaciones.push_back( &vaciarBaA ); nombres[vaciarBaA] = "Vaciar de B a A";
114    operaciones.push_back( &llenarA ); nombres[llenarA] = "Llenar A";
115    operaciones.push_back( &llenarB ); nombres[llenarB] = "Llenar B";
116    operaciones.push_back( &verterAllenandoB ); nombres[verterAllenandoB] = "Verter de A a B hasta llenar B";
117    operaciones.push_back( &verterBllenandoA ); nombres[verterBllenandoA] = "Verter de B a A hasta llenar A";
118    operaciones.push_back( &vaciarA ); nombres[vaciarA] = "Hechar todo A al piso";
119    operaciones.push_back( &vaciarB ); nombres[vaciarB] = "Hechar todo B al piso";
120
121    Estado inicial(0,0); // Al principio, ambos cántaros estan vacíos
122
123    cout << "\nPor preferencia en profundidad:\n";
124    mostrar_solucion( inicial, preferencia_profundidad( operaciones, inicial), nombres);
125    mostrar_estadisticas();
126
```

```
    cout << "Por profundidad limitada:\n";
    mostrar_solucion( inicial, profundidad_limitada(operaciones, inicial, 9), nombres );
    mostrar_estadisticas();

    cout << "\nPreferencia por amplitud:\n";
    mostrar_solucion( inicial, preferencia_amplitud( operaciones, inicial ), nombres );
    mostrar_estadisticas();

    cout << "\nProfundidad iterativa:\n";
    mostrar_solucion( inicial, profundidad_iterativa( operaciones, inicial, 9), nombres );
    mostrar_estadisticas();


    system("PAUSE");
}
```

6.3. laberinto.cpp

Usa punteros a funciones para encontrar la solucion a un laberinto hard-coded en el programa, demuestra el uso de la búsqueda bidireccional especificando los operadores inversos y dejando que el auxiliar `OperadoresInversos` los genere, tambien demuestra a `simplificar_bidireccional` para convertir una solución de dos caminos a una de un solo sentido.

De la STL usa `vector` y `pair`. Tambien la función auxiliar `make_pair` que permite construir un pair sin utilizar variables auxiliares.

Utiliza búsqueda bidireccional

```

#define IA_NO_CRONOMETRAR
#include "../include/ia_ciega.h"

using namespace std;
using namespace ia;


// se podría cargar el laberinto de un archivo de texto facilmente...
const unsigned int MAX_COLUMNAS = 73, MAX_FILAS = 47;
const unsigned int FILA_INICIO = 0, COLUMNA_INICIO = 1, FILA_FIN = 46, COLUMNA_FIN = 71;

const char* laberinto[] = {
"X XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"X       X       X     X     X         X X   X   X   X     X   X         X",
"XXX XXXXX XXX XXX XXX X XXX X XXX XXX X X X X X X X X XXX X X X XXXXX XXX",
"X   X     X X X   X   X X     X X   X X   X   X X X   X X   X   X   X   X",
"X XXX X XXX X X XXX XXX X XXXXX XXX X XXXXXXX X XXXXXXX XXXXXXXXX X XXX X",
"X X   X     X X   X     X       X   X   X X   X       X   X       X X   X",
"X X XXXXXXXXX XXX XXXXXXXXXXXXXXX XXXXX X X XXXXXXX X XXX X XXXXXXX X X X",
"X     X X     X           X   X   X X     X     X   X     X   X   X X X X",
"X XXX X X XXX XXXXXXXXXXXXX X X XXX X XXXXXXXXX X XXXXXXX XXX X XXX X X X",
"X   X   X   X       X     X X   X   X X       X X     X   X   X     X X X",
"XXXXXXXXXXX XXXXXXX X XXX X XXXXX XXX XXX XXX X XXXXX XXXXX XXXXX XXX XXX",
"X       X   X   X   X X X X         X   X   X   X X         X   X   X   X",
"X X XXX X XXXXX X XXX X X XXX XXXXX XXX XXXXX XXX X XXXXXXXXX X XXXXX X X",
"X X   X         X     X X     X       X         X X     X   X X X   X X X",
"X XXXXXXXXXXXXXXXXXXXXX XXXXX XXXXXXXXXXXXXXXXX X XXXXX X X X X X X X X X",
"X     X         X X   X X   X X     X       X X       X X X   X   X   X X",
"XXXXX XXX XXX X X X X X X X X X XXX X XXXXX X XXXXXXX X XXX XXXXXXXXXXX X",
"X     X   X   X     X   X X   X X     X     X   X   X X   X X         X X",
"X XXXXX XXXXXXXXXXXXXXXXXXXXXXX XXXXXXX XXXXXXX X X X XXX XXX X XXXXX X X",
"X X     X     X   X           X   X   X   X   X   X X   X X   X     X   X",
"X XXX XXX XXX X X X XXXXXXX X XXX X XXXXX X XXXXX X XXX X X XXXXXXX XXXXX",
"X   X   X X   X X X       X X   X X   X X   X   X X X X X     X   X X   X",
"XXX XXX X XXXXX X X XXXXX X XXX X X X X XXX X X X X X X XXXXX X X X XXX X",
"X       X X     X   X X   X   X X   X     X X X X X X X X   X   X X   X X",
"X XXXXX X X XXXXXXXXX X XXXXXXX XXXXXXXXX X X X X X X X X X X XXXXX X X X",
"X X   X X   X           X     X           X   X X X     X X X X   X X   X",
"X XXX XXXXXXX XXXXXXXXXXX XXX XXXXXXXXXXXXXXXXX XXXXXXXXX X XXX X X XXXXX",
"X   X         X X   X   X X X         X       X   X       X X   X   X   X",
```

```
"X X XXXXX XXXXX X X XXX X X X XXX XXXXX X XXXXX X X XXXXXXX X XXXXXXX X X",
"X X     X       X X     X X X X   X   X X     X X X   X   X X         X X",
"XXXXXXX XXXXXXX X XXXXXXX X X X XXX X XXX XXX X X XXX X X X XXXXXXXXXXX X",
"X       X         X   X   X   X     X   X X     X     X X   X X     X   X",
"X XXXXXXXXXXXXXXXXXXX X XXXXXXXXX XXXXX XXX XXXXXXXXXXX XXXXX X XXX X XXX",
"X                 X   X X   X   X     X     X     X   X       X X   X   X",
"XXXXXXX XXXXXXXXX X XXX X X X X XXXXXXXXXXXXX X XXX X XXX XXXXX X XXXXX X",
"X     X   X     X X   X   X   X         X   X X     X   X     X X X X   X",
"X XXX XXX X XXXXX XXX X XXXXXXXXXXXXX X XXX X XXXXXXXXX XXXXX X X X X X X",
"X   X X   X       X   X X       X   X X       X     X   X   X   X X X X X",
"XXX X XXXXX XXXXXXX XXXXX XXX X X X X XXXXXXXXXXXXX X X XXX XXXXX X X X X",
"X   X     X     X   X X   X   X X X X X   X   X   X   X   X     X   X X X",
"X XXXXXXX XXXXX X XXX X XXX XXX XXX X X X X X X X XXXXXXX X XXXXXXX X X X",
"X X           X     X   X X X     X X   X X X   X         X X     X X X X",
"X XXXXXXXXXXXXXXXXX X XXX X XXXXX X XXXXX X XXXXXXXXXXXXXXX X XXX XXX X X",
"X   X         X     X X   X     X X   X   X   X X         X     X   X X X",
"X X X XXXXXXX X XXXXX X X XXX X XXX X X XXXXX X X XXXXX X XXXXXXXXX X X X",
"X X   X         X     X X     X     X       X     X     X             X X",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX X"
};

struct Estado {
   //      fila     , columna
   pair<unsigned int, unsigned int> posicion;
   Estado() {}
   Estado( pair<unsigned int, unsigned int> pos ) : posicion(pos) {}
   bool operator<( const Estado& otro ) const {
      return posicion < otro.posicion;
   }
   friend ostream& operator<<( ostream& os, const Estado& e) {
      os << "(" << e.posicion.first << "," << e.posicion.second << ") ";
      return os;
   }
};

bool arriba( const Estado& antes, Estado& despues ) {
   if ( antes.posicion.first == 0 || laberinto[antes.posicion.first - 1][antes.posicion.second] != ' ' )
      return false;
   despues.posicion.first = antes.posicion.first - 1;
   despues.posicion.second = antes.posicion.second;
   return true;
}

bool abajo( const Estado& antes, Estado& despues ) {
   if ( antes.posicion.first == MAX_FILAS - 1 || laberinto[antes.posicion.first + 1][antes.posicion.second] != ' ' )
      return false;
   despues.posicion.first = antes.posicion.first + 1;
   despues.posicion.second = antes.posicion.second;
   return true;
}

bool izquierda( const Estado& antes, Estado& despues) {
   if ( antes.posicion.second == 0 || laberinto[antes.posicion.first][antes.posicion.second - 1] != ' ' )
      return false;
   despues.posicion.first = antes.posicion.first;
   despues.posicion.second = antes.posicion.second - 1;
   return true;
}

bool derecha( const Estado& antes, Estado& despues) {
   if ( antes.posicion.second == MAX_COLUMNAS - 1 || laberinto[antes.posicion.first][antes.posicion.second + 1] !=
      return false;
   despues.posicion.first = antes.posicion.first;
   despues.posicion.second = antes.posicion.second + 1;
   return true;
}

int main() {
   Operaciones<Estado> operaciones;
   NombresOperadores<Estado> nombres;
   OperadoresInversos< Operaciones<Estado> > inversas;
```

```
118
119    operaciones.push_back( &izquierda ); nombres[ &izquierda ] = "Izq"; inversas[ &izquierda ] = &derecha;
120    operaciones.push_back( &derecha ); nombres[ &derecha ] = "Der";
121    operaciones.push_back( &arriba ); nombres[ &arriba ] = "Arr"; inversas[ &arriba ] = &abajo;
122    operaciones.push_back( &abajo ); nombres[ &abajo ] = "Aba";
123
124    Estado inicial( make_pair(FILA_INICIO, COLUMNA_INICIO) );
125    Estado final( make_pair(FILA_FIN, COLUMNA_FIN) );
126
127    mostrar_solucion( inicial
128                      ,simplificar_bidireccional( profundidad_limitada_doble( operaciones, inicial, final, 500 ), i
129                      ,nombres );
130
131    /* Si usas MS Visual Studio .NET 2003, tendrás que usar estas cuatro lineas en vez de las anteriores tres.
132    pair< vector< Operaciones<Estado>::value_type >, vector< Operaciones<Estado>::value_type > > solucion;
133    solucion = profundidad_limitada_doble( operaciones, final, inicial, 500);
134    vector< Operaciones<Estado>::value_type> solucion_simple = simplificar_bidireccional( solucion, inversas.complem
135    mostrar_solucion( final, solucion_simple, nombres );*/
136
137    mostrar_estadisticas();
138 }
```

6.4. laberinto2.cpp

Usa punteros a funciones para encontrar la solucion a un laberinto ingresado por entrada estandar, poniendo la solucion en salida estandar. Demuestra el uso del contenedor `SecuenciaEstados` para convertir las operaciones devueltas como solucion en los estados intermedios

Una buena manera de usarlo es poner el laberinto en un archivo de texto. Tal archivo debe usar el caracter # para las paredes, la entrada del laberinto es el único espacio de la primera fila y la salida el único espacio en en la última fila. La primera y última columna consisten de solo paredes (#)

Si el laberinto esta en el archivo laberinto.txt y se quiere la solucion en el arcivo solucion.txt en la linea de comandos del sistema ponga:

```
1  c:\...> type laberinto.txt | laberinto2.exe > solucion.txt
```

El archivo de solucion consiste en el mismo laberinto pero con el caracter + marcando el camino.

```
1
4 #define IA_NO_CRONOMETRAR
5 #include "../include/ia_ciega.h"
6
7 using namespace std;
8 using namespace ia;
9
11 unsigned int MAX_COLUMNAS = 0, MAX_FILAS = 0;
12 unsigned int FILA_INICIO = 0, COLUMNA_INICIO = 0, FILA_FIN = 0, COLUMNA_FIN = 0;
13
14 typedef vector<string> Laberinto;
15 Laberinto laberinto;
16
17 ostream& operator<<(ostream& os, const Laberinto& lab) {
18    for( unsigned int i = 0; i < lab.size(); ++ i )
19       os << lab[i] << "\n";
20    os << "\n";
21    return os;
22 }
23
24 struct Estado {
25    //      fila     , columna
26    pair<unsigned int, unsigned int> posicion;
27    Estado() {}
28    Estado( pair<unsigned int, unsigned int> pos ) : posicion(pos) {}
29    bool operator<( const Estado& otro ) const {
30       return posicion < otro.posicion;
```

```
31    }
33    friend ostream& operator<<( ostream& os, const Estado& e) {
34       os << "(" << e.posicion.first << "," << e.posicion.second << ") ";
35       return os;
36    }
38    friend Laberinto& operator<<( Laberinto& lab, const Estado& e) {
39       lab[e.posicion.first][e.posicion.second] = '+';
40       return lab;
41    }
42 };
43
44 bool arriba( const Estado& antes, Estado& despues ) {
45    if ( antes.posicion.first == 0 || laberinto[antes.posicion.first - 1][antes.posicion.second] != ' ' )
46       return false;
47    despues.posicion.first = antes.posicion.first - 1;
48    despues.posicion.second = antes.posicion.second;
49    return true;
50 }
51
52 bool abajo( const Estado& antes, Estado& despues ) {
53    if ( antes.posicion.first == MAX_FILAS - 1 || laberinto[antes.posicion.first + 1][antes.posicion.second] != ' ' )
54       return false;
55    despues.posicion.first = antes.posicion.first + 1;
56    despues.posicion.second = antes.posicion.second;
57    return true;
58 }
59
60 bool izquierda( const Estado& antes, Estado& despues) {
61    if ( antes.posicion.second == 0 || laberinto[antes.posicion.first][antes.posicion.second - 1] != ' ' )
62       return false;
63    despues.posicion.first = antes.posicion.first;
64    despues.posicion.second = antes.posicion.second - 1;
65    return true;
66 }
67
68 bool derecha( const Estado& antes, Estado& despues) {
69    if ( antes.posicion.second == MAX_COLUMNAS - 1 || laberinto[antes.posicion.first][antes.posicion.second + 1] != '
70       return false;
71    despues.posicion.first = antes.posicion.first;
72    despues.posicion.second = antes.posicion.second + 1;
73    return true;
74 }
75
76 // Carga un laberinto de un archivo de texto. Es un poco larga porque verificamos que el archivo tenga el formato co
77 void cargar_laberinto() {
78    string linea, linea_anterior;
79    int cuenta = 0;
80    while ( getline(cin, linea) ) {
81       if ( ! cuenta ) { // si es la primera linea, buscamos el punto inicio
82          for( unsigned int c = 0; c < linea.size(); ++ c )
83             if ( linea[c] == ' ' )
84                COLUMNA_INICIO = c;
85       }else if ( linea.size() == 0 ){
86          while ( getline(cin,linea) )
87             if (linea.size() > 0 ) {
88                cerr << "ERROR: La linea " << cuenta + 1 << " esta en blanco.\n";
89                exit(1);
90             }
91          break;
92       }else if ( linea.size() != linea_anterior.size() ) {
93          cerr << "ERROR: La linea " << cuenta + 1 << " tiene una cantidad distinta de caracteres.\n" << "(anterior:"
94          exit(1);
95       }
96       laberinto.push_back( linea );
97       linea_anterior = linea;
98       cuenta++;
99    }
100    // buscamos el punto de la meta
101    for ( unsigned int c = 0; c < linea_anterior.size(); ++ c )
102       if ( linea_anterior[c] == ' ' ) {
103          COLUMNA_FIN = c;
```

```
104          break;
105       }
106    MAX_COLUMNAS = linea_anterior.size();
107    MAX_FILAS = cuenta;
108    FILA_FIN = MAX_FILAS - 1;
109    FILA_INICIO = 0;
110    if ( MAX_COLUMNAS < 3 || MAX_FILAS < 3 ) {
111       cout << "Es un laberinto demasiado pequeño!!";
112       exit(1);
113    }else if ( COLUMNA_INICIO == 0 || COLUMNA_FIN == 0 ) {
114       cout << "No se encontró el punto de partida o la meta";
115       exit(1);
116    }
117 }
118
119 int main() {
120    cargar_laberinto();
121
122    Operaciones<Estado> operaciones;
123    NombresOperadores<Estado> nombres;
124    OperadoresInversos< Operaciones<Estado> > inversas;
125
126    operaciones.push_back( &izquierda ); inversas[ &izquierda ] = &derecha;
127    operaciones.push_back( &derecha );
128    operaciones.push_back( &arriba ); inversas[ &arriba ] = &abajo;
129    operaciones.push_back( &abajo );
130
131    Estado final( make_pair(FILA_INICIO, COLUMNA_INICIO) );
132    Estado inicial( make_pair(FILA_FIN, COLUMNA_FIN) );
133
134    cerr << "Buscando...";
135    pair< vector< Operaciones<Estado>::value_type >, vector< Operaciones<Estado>::value_type > > solucion;
136    try {
137       // borland se cuelga con 1019 y da con 1018
138       solucion = profundidad_limitada_doble( operaciones, inicial, final, 6000);
139    }catch(...){
140       cerr << "Excepcion";
141    }
142    cerr << solucion.first.size() << "," << solucion.second.size() << "\n";
143    cerr << "Finalizado\n";
144    vector< Operaciones<Estado>::value_type> solucion_simple = simplificar_bidireccional( solucion, inversas.complem
145    cerr << "Se simplifico la solucion bi-direccional a uni-direccional:\n";
146    cout << solucion_simple.size() << "\n";
147
148    SecuenciaEstados<Estado, Operaciones<Estado> > estados_intermedios(inicial, solucion_simple );
149    cerr << "Se obtuvieron los estado intermedios\n";
150
151    mostrar_solucion( inicial, solucion_simple, nombres );
152    int x = 0;
153    for( SecuenciaEstados<Estado, Operaciones<Estado> >::iterator it = estados_intermedios.begin(); it != estados_in
154       cout << x << "-";
155       laberinto << *it;
156       cout << x ++ << " ";;
157    }
158    cerr << "Se anoto la solucion en el laberinto\n";
159    cout << "\n\n";
160    cout << laberinto;
161 }
162
```

6.5. pastor.cpp

Usa punteros a funciones para especificar las operaciones realizables sobre los estados. Demuestra cómo poner nombres a las operaciones que estan dadas como punteros a funciones.

Lo único que usa de la STL es el tipo `vector` y el `map`, sobre el vector solo usa el metodo `push_back` que sirve para aumentar un valor más al vector, y del `map` usa el operador corchetes (`operator[ ]`), que hace que un `map` se pueda usar más o menos como un vector.

Utiliza las búsquedas de preferencia por amplitud, profundidad limitada y profundidad iterativa.

Resuelve el problema del pastor, el lobo, la oveja y el pasto: El hombre (pastor), el lobo, la oveja y el pasto (forraje) están en la orilla izquierda del rio. El pastor tiene una canoa en que solo cabe él y una cosa más (ya sea el lobo, la oveja o el forraje) Si deja solos al lobo y a la oveja, muere la oveja; Si deja solos a la oveja y al forraje, se acaba el forraje. Debe llevar TODO, sano y salvo, a la otra orilla.

```
1 /* Resuelve el problema del pastor, el lobo, la oveja y el pasto:
2    El hombre (pastor), el lobo, la oveja y el pasto (forraje) están en la orilla izquierda del rio.
3    El pastor tiene una canoa en que solo cabe él y una cosa más (ya sea el lobo, la oveja o el forraje)
4    Si deja solos al lobo y a la oveja, muere la oveja;
5    Si deja solos a la oveja y al forraje, chau forraje.
6    Debe llevar TODO, sano y salvo, a la otra orilla.
7 */
8
9 #include <string>
10 #include <vector>
11 #include <iostream>
12
13 #include "../include/ia_ciega.h"
14
15 using namespace std;
16 using namespace ia;
17
18 enum Orilla {IZQ = 0, DER = 1};
19
20 struct Estado {
21    // Si hombre vale DER el hombre esta en la orilla derecha,
22    // si oveja vale IZQ la oveja esta en la orilla izquierda, etc.
23    Orilla hombre, lobo, oveja, forraje, bote;
24
25    Estado() {}
26    Estado( Orilla hombre_, Orilla lobo_, Orilla oveja_, Orilla forraje_, Orilla bote_ )
27    : hombre(hombre_), lobo( lobo_ ), oveja( oveja_ ), forraje( forraje_ ), bote( bote_ ) {
28    }
29    // Devuelve true ssi este estado es válido, es decir, ssi nadie se comerá a nadie
30    bool valido() const {
31       return !( ( lobo == oveja && hombre != lobo ) || (oveja == forraje && hombre != oveja) );
32    }
33    // Devuelve true ssi es un estado meta
34    bool es_meta() const {
35       return hombre == DER && lobo == DER && oveja == DER && forraje == DER
36             && bote == DER;
37    }
38    // Un operador de orden debil estricto, para que pueda estar en un std::set
39    bool operator<(const Estado& otro) const {
40       // formamos el patron de bits correspondiente a ambos estados y los comparamos
41       return ((((bote<<1 | forraje)<<1 | oveja)<<1 | lobo)<<1 | hombre)
42              < ((((otro.bote<<1 | otro.forraje  )<<1 | otro.oveja)<<1 | otro.lobo)<<1 | otro.hombre);
43    }
44    // Devuelve una cadena con la especificación del estado
45    // Solo es necesario si se quiere mostrar las soluciones con estados intermedios
46    friend ostream& operator<<(ostream& os, const Estado& e ) {
47       os << "(" << (e.hombre ? "Hd" : "Hi") << (e.lobo ? ", Ld" : ", Li") << (e.oveja ? ", Od" : ", Oi") << (e.forra
48       return os;
49    }
50 };
51
52
53 // El hombre solito a la derecha
54 bool h_der( const Estado& antes, Estado& despues) {
55    if ( antes.hombre == DER || antes.bote == DER )
56       return false;
57    despues = Estado( DER, antes.lobo, antes.oveja, antes.forraje, DER );
58    return despues.valido();
59 }
60
61 // El hombre solito a la izquierda
62 bool h_izq( const Estado& antes, Estado& despues) {
63    if ( antes.hombre == IZQ || antes.bote == IZQ )
```

```
64       return false;
65    despues = Estado( IZQ, antes.lobo, antes.oveja, antes.forraje, IZQ );
66    return despues.valido();
67 }
68
69 // el hombre y el lobo a la orilla derecha
70 bool hl_der(const Estado& antes, Estado& despues) {
71    if ( antes.hombre == DER || antes.lobo == DER || antes.bote == DER )
72       return false;
73    despues = Estado( DER, DER, antes.oveja, antes.forraje, DER );
74    return despues.valido();
75 }
76
77 bool hl_izq(const Estado& antes, Estado& despues) {
78    if ( antes.hombre == IZQ || antes.lobo == IZQ || antes.bote == IZQ )
79       return false;
80    despues = Estado( IZQ, IZQ, antes.oveja, antes.forraje, IZQ );
81    return despues.valido();
82 }
83
84 // El hombre y la oveja a la orilla derecha
85 bool ho_der(const Estado& antes, Estado& despues) {
86    if ( antes.hombre == DER || antes.oveja == DER || antes.bote == DER )
87       return false;
88    despues = Estado( DER, antes.lobo, DER, antes.forraje, DER );
89    return despues.valido();
90 }
91
92 bool ho_izq( const Estado& antes, Estado& despues) {
93    if ( antes.hombre == IZQ || antes.oveja == IZQ || antes.bote == IZQ )
94       return false;
95    despues = Estado( IZQ, antes.lobo, IZQ, antes.forraje, IZQ );
96    return despues.valido();
97 }
98
99 // El hombre y el forraje a la orilla derecha
100 bool hf_der( const Estado& antes, Estado& despues) {
101    if ( antes.hombre == DER || antes.forraje == DER || antes.bote == DER )
102       return false;
103    despues = Estado( DER, antes.lobo, antes.oveja, DER, DER );
104    return despues.valido();
105 }
106
107 bool hf_izq( const Estado& antes, Estado& despues) {
108    if ( antes.hombre == IZQ || antes.forraje == IZQ || antes.bote == IZQ )
109       return false;
110    despues = Estado( IZQ, antes.lobo, antes.oveja, IZQ, IZQ );
111    return despues.valido();
112 }
113
114
115 int main() {
116    Operaciones<Estado> operaciones;
117    NombresOperadores<Estado> nombres;
118    operaciones.push_back( &ho_der ); nombres[&ho_der] = "Hombre y oveja a la derecha";
119    operaciones.push_back( &ho_izq ); nombres[&ho_izq] = "Hombre y oveja a la izquierda";
120    operaciones.push_back( &hf_der ); nombres[&hf_der] = "Hombre y forraje a la derecha";
121    operaciones.push_back( &hf_izq ); nombres[&hf_izq] = "Hombre y forraje a la izquierda";
122    operaciones.push_back( &h_der ); nombres[&h_der] = "Hombre a la derecha";
123    operaciones.push_back( &h_izq ); nombres[&h_izq] = "Hombre a la izquierda";
124    operaciones.push_back( &hl_der ); nombres[&hl_der] = "Hombre y lobo a la derecha";
125    operaciones.push_back( &hl_izq ); nombres[&hl_izq] = "Hombre y lobo a la izquierda";
126
127    Estado inicial(IZQ,IZQ,IZQ,IZQ,IZQ); // Todos en la orilla izquierda
128
129    mostrar_solucion(inicial, preferencia_amplitud(operaciones, inicial), nombres );
130    mostrar_estadisticas();
131
132    mostrar_solucion(inicial, profundidad_limitada(operaciones, inicial, 18 ), nombres );
133    mostrar_estadisticas();
134
```

```
135    mostrar_solucion(inicial, profundidad_iterativa(operaciones, inicial, 18 ), nombres );
136    mostrar_estadisticas();
137
138    system("PAUSE");
139 }
```

6.6. reinas.cpp

Usa functores (objetos-función) para especificar las operaciones sobre los estados. También demuestra como utilizar el parámetro opcional `registrar_solucion` para encontrar todas las soluciones al problema.

De la STL usa `vector` y `pair` (pair es una estructura simple con dos miembros de, el primero se llama `first` y el segundo `second`)

Utiliza la búsqueda por profundidad limitada

```
1 /* Resuelve el problema de las ocho reinas, mostrando todas las soluciones posibles:
2    Colocar ocho reinas en un tablero de ajedrez de manera que ninguna este amenazada por otra.
3    En vez de una función por operador, se usa un objeto función en cuyo constructor se establece
4    en que fila tiene que poner la reina.
5 */
6 #include <vector>
7 #include <algorithm>
8 #include <iostream>
9 #include <string>
10 #include <sstream>
11 #include <cassert>
12
13 //#define IA_NO_CRONOMETRAR
14
15 #include "../include/ia_ciega.h"
16
17 using namespace std;
18 using namespace ia;
19
20 // Tamaño del tablero (= cantidad de reinas)
21 const unsigned int TAM = 8;
22
23 // v[3].first = fila donde está la 4ta reina, v[3].second = columna donde esta la 4ta reina
24 typedef vector< pair<unsigned int, unsigned int> > VectorReinas;
25
26 // Este es el estado
27 struct Tablero {
28    // La posición en que está cada reina
29    VectorReinas reinas;
30    Tablero() {}
31    bool es_meta() const {
32       return reinas.size() == TAM;
33    }
34    bool operator<(const Tablero& otro) const {
35       return lexicographical_compare( reinas.begin(), reinas.end(), otro.reinas.begin(), otro.reinas.end() );
36    }
37    friend ostream& operator<<(ostream& os, const Tablero& t ) {
38       os << "[ ";
39       for ( VectorReinas::const_iterator r = t.reinas.begin(); r != t.reinas.end(); ++ r )
40          os << "(" << r->first << "," << r->second << ") ";
41       os << "]";
42       return os;
43    }
44 };
45
46 // Pone una reina en la fila dada a su constructor, si ya hay, trata de moverla hacia la derecha
47 class PonerReina : public Operacion<Tablero> {
48 public:
49    unsigned int fila; // Fila en la que se debe poner la reina
50    PonerReina(const unsigned int& fila_) : fila( fila_ ) {
51       ostringstream os; os << "Mover reina fila: " << fila; nombre = os.str();
52    }
```

```
53    // Mueve la fila-sima reina al siguiente cuadrado de la derecha
54    bool operator()(const Tablero& antes, Tablero& despues ) const {
55       if ( antes.reinas.size() == fila ) {
56          despues.reinas = antes.reinas;
57          despues.reinas.push_back( make_pair(fila,0) );
58       }else if ( antes.reinas.size() == fila + 1) {
59          despues.reinas = antes.reinas;
60          despues.reinas.back().second ++;
61       }else
62          return false;
63       // Ir recorriendo la ultima reina a la derecha hasta encontrar una casilla no amenazada
64       bool amenazada;
65
66       pair<unsigned int, unsigned int>& ultima = despues.reinas.back();
67       do {
68          amenazada = false;
69          for ( VectorReinas::const_iterator reina = despues.reinas.begin(); reina != despues.reinas.end() - 1;  ++ r
70             if ( reina->second == ultima.second // misma columna
71                   || reina->second - reina->first == ultima.second - ultima.first //misma diagonal
72                   || ( TAM - reina->second) - reina->first == ( TAM - ultima.second) - ultima.first ) { // misma dia
73                amenazada = true;
74                ultima.second ++;
75                break; // salimos del ciclo for
76             }
77       }while( amenazada && ultima.second < TAM);
78       return ultima.second < TAM && ! amenazada;
79    }
80 };
81
82 typedef vector< PonerReina* > Ruta;
83
84 vector<Tablero> estados_meta;
85
86 bool registrar(const Ruta&, const Tablero& tablero) {
87    estados_meta.push_back( tablero );
88    return true;
89 }
90
91 int main() {
92    typedef vector< PonerReina* > Operaciones;
93    Operaciones operaciones;
94    for ( int fila = 0; fila < TAM; ++ fila )
95       operaciones.push_back( new PonerReina( fila ) );
96
97    Tablero inicial;
98
99    cout << "\nPor profundidad limitada:\n";
100
101    profundidad_limitada( operaciones, inicial, 25, &registrar);
102
103    mostrar_estadisticas();
104    cout << "\nSe encontraron " << estados_meta.size() << " soluciones:\n";
105    for( unsigned int i = 0; i < estados_meta.size(); ++ i )
106       cout << estados_meta[i] << "\n";
107    estados_meta.clear();
108
109    // Liberar la memoria del vector de operaciones
110    for ( int fila = 0; fila < TAM; ++ fila )
111       delete operaciones[fila];
112 }
```

6.7. viajero.cpp

Demuestra el uso de `primero_mejor` y un functor (`Elige_mejor_ciudad`) para hallar el camino entre dos ciudades. Las operaciones son functores (objetos-función) instancias de la clase Viajar.

Ademas de la búsqueda de primero el mejor, usa las búsquedas por profundidad limitada y preferencia amplitud.

```
1
```

```
3 #include <algorithm>
4 #include <iostream>
5 #include <iomanip>
6 #include <string>
7 #include <cassert>
8 #include <map>
9 #include <set>
10
11 #define IA_NO_CRONOMETRAR
12 #include "../include/ia_ciega.h"
13
14 using namespace std;
15 using namespace ia;
16
17 const string ORIGEN = "Tarija";
18 const string DESTINO = "Pando";
19
20 // Destinos son parejas de ciudad colindante y la longitud de la carretera a ella
21 typedef map<string, int> Destinos;
22 // a cada ciudad se le asocia un conjunto de destinos
23 typedef map< string, Destinos > Mapa;
24 Mapa mapa;
25
26 // A cada ciudad se le asocia su distancia lineal a la ciudad meta
27 Mapa distancias;
28
29 // Esta es la estructura estado, simplemente indica en qué ciudad estamos
30 struct Ciudad {
31    string nombre;
32    bool operator<( const Ciudad& otro ) const {
33       return nombre < otro.nombre;
34    }
35    bool es_meta() const {
36       return nombre == DESTINO;
37    }
38    friend ostream& operator<<(ostream& os, const Ciudad& ciudad ) {
39       os << "(" << ciudad.nombre << ") ";
40       return os;
41    }
42 };
43
44 // Esta clase hace las operaciones, o sea, los viajes
45 struct Viajar : public Operacion<Ciudad> {
46    string destino; // destino del viaje
47    Viajar() {}
48    Viajar(string destino_) : destino( destino_ ) {
49    }
50    bool operator()(const Ciudad& antes, Ciudad& despues) const {
51       // verificamos que hay carretera a la ciudad a la que nos toca probar
52       if ( mapa[antes.nombre].find( destino ) == mapa[antes.nombre].end() )
53          return false;
54       despues.nombre = destino;
55       return true;
56    }
57 };
58
59 // Este es el functor que decide qué ciudad elegir
60 struct Elige_mejor_ciudad {
61    bool operator()(const pair<Ciudad, vector<Viajar*> >& izq, const pair<Ciudad, vector<Viajar*> >& der) const {
62       // preferimos la ciudad cuya distancia lineal al destino sea menor
63       return distancias[DESTINO][izq.first.nombre] <= distancias[DESTINO][der.first.nombre];
64    }
65 };
66
67 // Para mostrar los mapas creados, realmente no es necesario...
68 void mostrar_mapa( Mapa& mapa ) {
69    cout << "-----------\n";
70    for( Mapa::iterator it_partida = mapa.begin(); it_partida != mapa.end(); ++ it_partida )
71          for( Destinos::iterator it_destino = it_partida->second.begin()
72                                                   ; it_destino != it_partida->second.end(); ++ it_destino )
73             cout << "De " << it_partida->first << " a " << it_destino->first << " hay "
```

```
74                    << it_destino->second << "kms\n";
75
76 }
77
78 int main() {
79    // a cada origen vs. destino se le asigna la distancia
80    mapa["Pando"]["La Paz"] = 540;
81    mapa["Pando"]["Beni"] = 430;
82    mapa["Beni"]["La Paz"] = 610;
83    mapa["Beni"]["Santa Cruz"] = 640;
84    mapa["La Paz"]["Oruro"] = 229;
85    mapa["Santa Cruz"]["Cochabamba"] = 540;
86    mapa["Santa Cruz"]["Chuquisaca"] = 680;
87    mapa["Cochabamba"]["Oruro"] = 228;
88    mapa["Cochabamba"]["Potosi"] = 532;
89    mapa["Cochabamba"]["Chuquisaca"] = 366;
90    mapa["Oruro"]["Potosi"] = 335;
91    mapa["Potosi"]["Chuquisaca"] = 166;
92    mapa["Potosi"]["Tarija"] = 334;
93    mapa["Chuquisaca"]["Tarija"] = 549;
94    // Se generan los caminos inversos
95    for( Mapa::iterator it_partida = mapa.begin(); it_partida != mapa.end(); ++ it_partida )
96       for( Destinos::iterator it_destino = it_partida->second.begin(); it_destino != it_partida->second.end(); ++ it
97          mapa[ it_destino->first ][it_partida->first] = it_destino->second;
98
99    mostrar_mapa(mapa);
100
101    // Cargar las distancias lineales hasta la ciudad de pando
102    distancias["Pando"]["La Paz"] = 500;
103    distancias["Pando"]["Beni"] = 300;
104    distancias["Pando"]["Oruro"] = 600;
105    distancias["Pando"]["Santa Cruz"] = 800;
106    distancias["Pando"]["Cochabamba"] = 500;
107    distancias["Pando"]["Potosi"] = 1000;
108    distancias["Pando"]["Chuquisaca"] = 1100;
109    distancias["Pando"]["Tarija"] = 1200;
110
111    // Creamos las operaciones posibles, esto es, todos los destinos posibles
112    vector<Viajar*> operaciones;
113    for( Mapa::iterator it_ciudad = mapa.begin(); it_ciudad != mapa.end(); ++ it_ciudad )
114       operaciones.push_back( new Viajar(it_ciudad->first) );
115
116    // Por fin empezamos la búsqueda...
117    Ciudad inicial;
118    inicial.nombre = ORIGEN;
119    mostrar_solucion( inicial, profundidad_limitada(operaciones, inicial, 15) );
120    mostrar_solucion( inicial, primero_mejor<Elige_mejor_ciudad>( operaciones, inicial ) );
121    mostrar_solucion( inicial, preferencia_amplitud( operaciones, inicial) );
122 }
```

7. IA Ciega 0.9 Documentación de páginas

7.1. Uso general

Las funciones de búsqueda estan en el namespace ia y son así: (las que estan en el namespace ia::detalle son internas y no es necesario llamarlas directamente)

```
template< [...] >
nombre_funcion(const Operaciones_t &operaciones, const Estado_t &inicial, [...], (bool*)registrar_solucion( [...] ) );
```

No te preocupes por lo de `template`, para llamar a las funciones no necesitas especificar ninguno de esos tipos, eso se hará automáticamente (excepto con Microsoft VC Toolkit y VC NET 2003). El primer parámetro siempre es una Lista de operaciones y el segundo un Estado inicial.

7.1.1. Lista de operaciones

Las funciones de búsqueda reciben siempre como primer parámetro un contenedor secuencial de las operaciones que se pueden aplicar a los estados para expandirlos. Esta secuencia puede ser una secuencia (p.e. un `vector`) de punteros a las funciones que se pueden usar para hacer expansiones, también puede ser una secuencia (p.e. un `vector`) de punteros a una clase base (normalmente `ia::Operacion<Estado>`) cuyos herederos definen al operador de paréntesis (`operator()`). Ten en cuenta que el uso de una clase base para las operaciones implica una llamada a una función virtual (al `operator()`) y esto equivale a dos indirecciones, es preferible utilizar una sola clase cuyas distintas instancias ejecuten de distinta manera a su `operator()`

7.1.1.1. Operaciones como funciones La lista de operaciones para expandir estados puede ser una secuencia de punteros a funciones (en este caso las funciones son `mi_operacion1` y `mi_operacion2`), así:

```
#include "ia_ciega.h"
using namespace std;
struct Estado {
   // Después se indica cómo debe ser esta estructura ...
};

bool mi_operacion1( const Estado& antes, Estado& despues ) {...}
bool mi_operacion2( const Estado& antes, Estado& despues) {...}
// ... etc
int main() {
    Operaciones<Estado> operaciones;
    NombresOperadores<Estado> nombres;
    operaciones.push_back( &mi_operacion1 );
    nombres[ &mi_operacion1] = "Mi operacion 1";
    operaciones.push_back( &mi_operacion2 );
    nombres[ &mi_operacion2] = "Mi operacion segunda";
    //...
    // Establecer el estado inicial
    Estado inicial(3,4);
    // llamar a la función de búsqueda deseada...
    mostrar_solucion( inicial, profundidad_iterativa( operaciones, inicial, 11), nombres );
    // etc...
}
```

Fijate cómo usamos las estructuras auxiliares `Operaciones` y `NombresOperadores` dando el parámetro Estado.

Cada una de las funciones mi_operacion1, mi_operacion2, etc, reciben como primer estado el estado actual y deben poner en el segundo parametro es estado resultante de aplicar la operacion. Si no se puede aplicar la operación debe devolver falso, si se puede aplicar debe devolver verdadero. Cuando devuelve falso, no se usará el valor del segundo parámetro fuera de esta funcion, así que se puede dejar como basura.

7.1.1.2. Operaciones como clases (functores) Existen dos maneras de trabajar: definir una clase distinta para cada operación (considere usar punteros a funciones), la segunda es la que se explicará aquí, a saber, una sola clase para todas las operaciones, el comportamiento específico de la operación se establece en el constructor de la clase.

La lista de operaciones para expandir estados puede ser una secuencia de punteros a una clase. En la lista de operaciones se ponen punteros objetos de esta clase, cada instancia ejecuta el `operator()` de acuerdo a la operacion que realiza, así:

```
struct Estado {
   // Después se indica cómo debe ser esta estructura...
};

class Accion : public Operacion<Estado> {
   ... // aquí puedes poner los miembros datos de la operacion
public:
   Accion(...) {...}
   bool operator()(const Estado_t& antes, Estado_t& despues) const {
      ... // Aplica la operacion de acuerdo a sus miembros dato
   }

}
```

En el constructor se ajustan los miembros datos del objeto, que son los parámetros que definen cómo se realizará la operación

La clase base utilizada `Operacion<Estado>` está definida en esta librería y contiene un miembro dato `nombre` en el que se puede poner el nombre de la operación, así como debería ser mostrada en la solución, alternativamente se puede re-implementar la funcion `get_descripción()` para mostrar la operacion. Todo esto es opcional y solo es requerido si se utiliza la función mostrar_solucion de esta libreria.

No es necesario heredar de `Operacion<Estado>`; pero, se herede o no, se debe declarar el `operator()` de esta manera en la clase base (en Operacion<Estado> ya esta declarada), y/o implementarla en cada clase (heredera):

```
virtual bool operator()(const Estado_t& antes, Estado_t& despues) const
```

este operador debe cumplir con los requisitos especificados anteriormente para las operaciones como funciones, con el requerimiento adicional de que no deben modificar los miembros datos del objeto (por eso es un método `const`). Si crees que necesitas cambiar algun miembro dato ten en cuenta que el orden en que se llama al `operator()` es indefinido, (para que se pueda cambiar un miembro dato `m` de tipo `Tipo` este debería ser declarado como `mutable`, así: `mutable Tipo m`;)

Las funciones de búsqueda solo utilizaran los objetos pasados en la lista de operaciones, no crearán nuevas instancias.

El ejemplo de las 8 reinas (reinas.cpp) hace uso de esta técnica, dandole en el constructor la fila en que debe poner una reina cada instancia de una misma clase.

7.1.2. Estado inicial

El segundo parámetro de las funciones de búsqueda es el estado inicial desde el que se empieza la búsqueda.

Un estado es una estructura (o clase) con esta interfaz:

```
struct Estado {
  // ... aquí puedes poner los atributos de tu estado
  Estado() {...} // Un constructor por defecto, Normalmete no hará nada
  ... // aquí puedes poner otros constructores
  // Debe devolver true ssi es un estado meta
  bool es_meta() const {
     ...
  }
  // Un operador menor que, de orden debil estricto, para que pueda estar en un std::set
  bool operator<(const Estado& otro) const {
     ...
  }
  // Devuelve una cadena con la especificación del estado (opcional)
  friend ostream& operator<<(ostream& os, const Estado& e ) {
     os << ... ;
     return os;
  }
};
```

El constructor por defecto y todas estos métodos (excepto el `operator<<`) deberían terminar rápido porque son llamados varias veces durante las búsquedas. Los estados construidos con el constructor por defecto se usan solamente para ser el segundo parámetro de las funciones que se usan para expandir estados (o del `operator()` si son functores) así que su inicialización puede hacerse en la función u `operator()`,

`es_meta()` se llama en todos los estados generados.

`operator<` se usa para poner al estado en un `std::set` de estados visitados, debe ser un strict weak ordering como se define en la STL, es decir que si a $<$ b entonces es falso que b $<$ a, y si a $<$ b y b $<$ c entonces a $<$ c. Dos estados A y B se considerarán equivalentes (es decir solo se explorará uno) si es falso que A $<$ B y es falso que B $<$ A.

`operator<<` se usa en mostrar_solucion con dos parámetros, así que no afecta a la velocidad de las funciones de búsqueda

7.1.3. Uso del parámetro registrar_solucion

Los algorítmos de búsqueda reciben, opcionalmente, un parámetro registrar_solucion que es un puntero a una función que se llamará cada vez que se encuentre una solución. Es una función "callback" aquí se describe cómo se usa.

La funcion puede ser así

```
bool registrar_solucion(const vector<X>& ruta, const Estado& estado)
```

donde X es el tipo de las operaciones puestas inicialmente en el contenedor de operaciones que se pueden aplicar (que pueden ser punteros a objetos funcion (o sea, a functores) o punteros a funciones).

Todas las búsquedas terminarán al encontrar la primera solución, si desea continuar la búsqueda de más soluciones debe implementar esta función y pasar un puntero a ella a la función de búsqueda que use, la búsqueda llamará a su función cada vez que halle una solución. Si desea que la búsqueda continue devuelva true, si desea terminarla devuelva false en este último caso la función de búsqueda originalmente llamada devolverá la solución que se acaba de hallar; por otra parte, si su función registrar_solucion siempre devuelve true eventualmente la función de búsqueda terminará devolviendo una ruta vacía.

En el primer parámetro recibirá un vector con punteros a las operaciones que se hicieron para encontrar la meta recién hallada; en el segundo parámetro estará el estado meta que se halló.

Por ejemplo, si las operaciones se dieron usando el tipo auxiliar Operaciones<Estado> (o sea, usando punteros a funciones) entonces ruta sera de tipo `vector<bool(*)` (const Estado&, Estado&)> , es decir un vector de punteros a las funciones que se aplicaron al estado inicial para llegar al estado meta dado como segundo parámetro.

Vaya a la pestaña Módulos y elija Algoritmos de búsqueda para obtener ayuda sobre una función de búsqueda en particular

7.2. Consejos prácticos

Recuerda poner `using namespace ia`, o empezar todos los usos de identificadores de la libreria con `ia::`

Es muy importante que las funciones de operaciones o los métodos operator() reciban el primer parámetro como "const Tipo& antes" y el segundo como "Tipo& despues" donde Tipo es el tipo de los estados. No se debe modificar el primer parámetro!.

La clase (o estructura) que uses como Estado debe ocupar poca memoria, su constructor por defecto debe ser rápido, preferiblemente vacío.

Las funciones para aplicar operaciones deben ser rapidísimos. (Si utilizas objetos-funcion, el `operator()` debe ser rapidísimo)

Si utilizas functores, deben ocupar poca memoria. Si necesitas muchos datos en los functores considera ponerlos en memoria global y mantener solo una referencia a esos datos en los functores.

7.3. Uso de macros para deshabilitar estadísticas y registro de soluciones.

La librería puede obtener algunas métricas sobre la cantidad de estados visitados, expandidos, etc esto reduce un poco la velocidad de la búsqueda.

Para evitar el cálculo de métricas ponga la linea:

```
#define IA_NO_ESTADISTICAS
```

antes de # incluir esta librería.

Para evitar que se tomen métricas de tiempo de ejecución ponga la línea

```
#define IA_NO_CRONOMETRAR
```

antes de # incluir esta ibrería. Es necesario definir esta macro si no está compilando en plataforma Windows.

La precisión de las métricas de tiempo tienen un error de (+/-)8 milisegundos

Si se definen ambas macros el cálculo de estadísticas no se llevará a cabo y las búsquedas se harán un poco mas rápido, la definición de solo una también aumenta la velocidad.

Las funciones de búsqueda reciben un parámetro opcional `registrar_solucion` que puede ser llamado cada vez que se halla una solucion y permite continuar la busqueda para encontrar todas las soluciones; para deshabilitar esta característica (y así aumentar un poco la velocidad de los algorítmos) ponga la linea:

```
#define IA_NO_REGISTRAR_SOLUCION
```

antes de # incluir esta librería. De todas maneras se podrá pasar el parámetro registrar_solucion a las funciones, pero no se utilizará, esto es así para que el programador usuario no deba modificar su propio código fuente.

7.4. Rationale

Se usan otras funciones para llamar a las que estan en el namespace ia::detalle porque las que estan en detalle necesitan más parámetros que los que el llamador quisiera llenar con información de su propio problema, por otra parte el paso de parámetros en la primera llamada a las funciones de detalle deben cumplir varios requerimientos que al llamador solamente le dificultarian su uso.

Se prefiere usar plantillas y no usar funciones virtuales para la realización de operaciones y la comparación de estados porque una llamada a una función virtual es un poco mas lenta que la llamada a funciones no-virtuales. Esto tiene más importancia porque la función virtual sería llamada en un ciclo interno durante las búsquedas

Se usa el lenguaje C++ porque tiene las siguientes características necesarias para esta librería, y no se encuentran en otros lenguajes: 1) Plantillas, no solo a nivel de contenedores, si no de operadores, por ejemplo para usar la misma función con functores y funciones. 2) Preprocesador, para poder deshabilitar capacidades y así ganar velocidad. 3) Generación de código eficiente, debido a que el código es ejecutado directamente por el hardware. 4) Librerias estandarizadas con variedad de estructuras de datos, algoritmos ortogonales a las estructuras de datos.

Se utilizan macros del preprocesador para deshabilitar el cálculo de estadísticas y registro de soluciones porque de esta forma una vez deshabilitadas permitirán un funcionamiento tan rápido como si nunca hubiera habido posibilidad de utilizar tales características. (No es necesario evaluar un booleano para tomar la decisión de usarlas).

Para ver detalles específicos de la implementacion consulte la siguiente sección Detalles de implementacion y el código fuente, que tiene muchos comentarios explicativos que no aparecen en este manual.

7.5. Preguntas Hechas Frecuentemente

(En realidad nadie hizo preguntas, pero estas respuestas dan información importante que no cabe en otros sitios de este manual)

- *¿No sería más fácil hacer mi propia función de búsqueda en vez de aprender a usar toda esta librería?* . No creo. Son algoritmos difíciles de implementar, mas bíen sería bueno empezar usando esta librería (u otra) y, después de comprobar que tus estados y operaciones funcionan, crear tus propias funciones de búsqueda. No tienes que aprender a usar toda la libreria, por ejemplo si no has usado mucho functores o la STL puedes limitarte a usar una función por cada operación.
- *¿Tengo que leer todo el código fuente para aprender a usar esta librería?* No, basta con leer los comentarios del código fuente, estos se presentan en forma más legible en la documentacion HTML incluida en el CD de la librería.
- *He visto el programa fuente y se usan características avanzadas de C++ (plantillas, STL, functores, etc.) ¿Necesito aprender todo eso para usar esta librería?* No. Mas bien hay que empezar por los ejemplos. Especialmente el de los cántaros, el laberinto y el pastor usan muy poco de la STL y lo mínimo de plantillas, si tienes problemas con eso te aconsejo hacer uso de la libreria empezando sobre uno de los ejemplos, en vez de empezar en blanco.
- *¿Puedo usar este programa en un producto comercial?* Si. Pero si tu producto incluye el código fuente tienes que incluir el código fuente de esta librería y también el archivo licencia_1_0.txt en la instalación y no borrar mi nombre del código fuente (tu producto puede utilizar otra licencia, la mia solo hara referencia a mi porción de código). Puedes incluir tu nombre en ia_ciega.h si añadiste algo a la librería, pero en ningún caso debes hacer creer que tú hiciste el código que yo hice ¿Obvio no?. Si tu producto está solo en forma binaria (sin código fuente) no es necesario que incluyas el archivo de licencia ni que aparezca mi nombre, una pequeña mención en "Acerca de" sería un bonito detalle de tu parte, tampoco debes hacer creer que tú hiciste la parte que yo hice.

7.6. Detalles de implementacion

En esta página se explican detalles útiles solamente a aquellos que quieran modificar o mejorar esta librería, o quienes esten interesados en sus detalles.

Se supone un conocimiento práctico de los contenedores estandar de C++ (parte de la STL) y una lectura al resto de este documento, especialmente la seccion Rationale.

La librería esta dividida en tres partes conceptuales: la interfaz, los algoritmos y los auxiliares (tanto estructuras de datos como funciones).

1. La **interfaz** comprende todo aquello a lo que el programador usuario puede acceder, esto es, funciones y tipos que se encuentran en el namespace ia, se puede usar todas las capacidades de esta libreria con solo ese conocimiento. Pertenecen a esta parte ia::profundidad_limitada_doble, ia::profundidad_iterativa, ia::mostrar_solucion, el tipo ia::Operacion, etc.
2. Los **algoritmos** se encuentran ocultos en el namespace ia::detalle y hay uno por cada método de búsqueda que aparece en la interfaz. Pertenecen a esta parte ia::detalle::profundidad_limitada_doble, ia::detalle::profundidad_iterativa
3. Los **auxiliares** tambien estan en el namespace ia::detalle y son utilizados tanto por los algoritmos como por la interfaz. Tenga en cuenta que en el resto de este documento se ha usado el término auxiliares haciendo referencia a ciertas funciones/tipos de la *interfaz* que no pertenecen a esta categoría conceptual.

7.6.1. Detalles sobre la interfaz

El uso de la interfaz se ha explicado en el resto de este documento, aquí se explica cómo esta implementada.

Las funciones de búsqueda de la interfaz tienen el propósito de que los algoritmos reciban la menor cantidad de parámetros posibles, y permitir que el compilador elija el tipo de los parámetros de plantilla necesarios, así como a adaptar las llamadas de acuerdo a la macro IA_NO_REGISTRAR_SOLUCION.

Tómese el código de la funcion de interfaz preferencia_amplitud como ejemplo

```
template <typename Estado_t, typename Operaciones_t>
vector< typename Operaciones_t::value_type> preferencia_amplitud(const Operaciones_t& operaciones
            , const Estado_t& inicial
            , bool(*registrar_solucion)( const vector<typename Operaciones_t::value_type>&
                                        ,const Estado_t& ) = NULL ) {
   typedef vector< typename Operaciones_t::value_type > Ruta_t;
   IA_REINICIAR_ESTADISTICAS;
   detalle::EstadoYRuta<Estado_t, Operaciones_t> principio;
   principio.push_back( make_pair( inicial, Ruta_t() ) );

   IA_INICIO_CRONOMETRO;
```

En esas lineas, en base a los parámetros recibidos se inicializan las variables requeridas por las funciones de algoritmos, se inician los contadores estadísticos y el cronometraje

```
   if ( inicial.es_meta() ) {// El algoritmo interno no verifica que el estado inicial sea el estado meta
      IA_IF_DEVUELVE_FALSO_REGISTRAR_SOLUCION( Ruta_t(), inicial )
         ; // no-operacion
      return vector<typename Operaciones_t::value_type>();
   }
```

La macro IA_IF_DEVUELVE_FALSO_REGISTRAR_SOLUCION ejecuta a la funcion apuntada por registrar_solucion si esto esta habilitado, de otra forma no hace nada.

En caso de que el estado inicial sea el estado meta, y el algoritmo interno no revise esta situacion con el estado inicial, son las funciones de la interfaz las que se encargan de llamar a la funcion registrar_solucion, como se ve en el ejemplo.

```
   Ruta_t solucion = detalle::preferencia_amplitud<Estado_t, Operaciones_t>(operaciones, principio
#ifndef IA_NO_REGISTRAR_SOLUCION
                                                            , registrar_solucion
#endif
                                                            );
   IA_FIN_CRONOMETRO;
   return solucion;
}
```

finalmente llama a la función de algoritmo interna con los parámetros que ella requiere, usando al preprocesador envian el parámetro registrar_solucion solamente si no esta definida la macro IA_NO_REGISTRAR_SOLUCION. Termina por devolver lo mismo que la función interna.

7.6.2. Detalles sobre la implementación

Siempre se ha minimizado el uso de variables locales en los algoritmos recursivos.

Internamente se utiliza el contenedor vector<> para almacenar datos que no se necesita ordenar, porque todas las inserciones y borrados se realizan al final y en esto es el más eficiente, salvo cuando se necesita aumentar su capacidad interna, sin embargo esto ocurre rara vez. La otra alternativa hubiera sido el uso de list, pero internamente es una lista *doblemente* enlazada lo cual hace que las inserciones y eliminaciones sean más lentas que en un vector. (Los vectores de la STL cambian de tamaño dinámicamente pero no se encuentran en el heap). Se utilizan set y multiset cuando se necesita que los elementos esten ordenados.

Es un hecho que un código eficiente en memoria o tiempo tiende a ser menos legible, se ha intentado mejorar la legibilidad a lo largo de la libreria, sin embargo sí se han usado ciertas expresiones que ameritan explicación:

Una técnica usada a lo largo de la librería para evitar el copiado de datos de contenedores es la siguiente, en vez de hacer

```
contenedor1 = contenedor2;
```

se ha utilizado el método swap de uno de los contenedores, de esta manera

```
contenedor1.swap( contenedor2 );
```

esto es menos legible y puede parecer menos eficiente ya que equivaldría a:

```
aux = contenedor1;
contenedor1 = contenedor2;
contenedor2 = aux;
```

Sin embargo la especificación de la STL indica que lo que en realidad se intercambia con el método swap no son los elementos de los contenedores, sino simplemente los iteradores de principio y fin de ambos (es decir seis asignaciones de tipos atómicos en vez de copiar todos los elementos de un contenedor a otro). El hecho de que el segundo contenedor quede con los datos del primero resulta ser irrelevante debido a que no se vuelve a utilizar tal contenedor en los algoritmos, o porque lo que corresponde, en los algoritmos, es vaciarlo. La alternativa hubiera implicado: 1) que en algun momento se hubiera tenido una copia de un contenedor, ocupando el doble de memoria de la requerida. 2) se hubiera requerido que los estados necesiten un copy-constructor.

Esta otra expresión tampoco es muy clara, se usa para ejecutar una operacion a un estado para obtener el estado sucesor:

```
if ( ! (*(*operacion))(estado, resultante) )
    ...;
```

Esto es lo que hace posible que la misma función funcione con functores y funciones, en el caso de un contenedor de punteros a funciones implica la dereferencia de un iterador (llamado 'operacion'), de un puntero a función y la llamada a la función apuntada, en el caso de un contenedor de punteros a functores implica la dereferencia de ese mismo iterador, de un puntero a un objeto y la llamada al `operator()` del objeto apuntado. En ambos casos la negacion implica que el cuerpo del `if` se ejecutará si NO es posible realizar la operación en el estado dado, en caso contrario el estado resultante estará almacenado en `resultante`.

Esta otra expresion se usa para notificar a la funcion callback de una solucion hallada:

```
if ( ! registrar_solucion || ! (*registrar_solucion)(ruta, actual) )
    ...;
```

Aprovechamos el short-circuit de las expresiones booleanas de C++ para invocar al callback solamente si no es nulo y realizar el cuerpo del if solamente si no había funcion callback o habiendo sido ejecutada devolvió falso.

7.6.3. Una nota sobre el estilo

El código fuente debe ser legible. Muchas lineas de código exceden el ancho de la pantalla (o de la página) se ha preferido cortar las lineas bajo esta premisa básica: una linea que lógicamente debería ser parte de la anterior SIEMPRE empezará con un símbolo de puntuación. Aunque esto no es lo que más a menudo se usa en otros programas, ayuda mucho a entender rápidamente el código, comparese este pedazo de código, que sigue el esquéma típico (no usado en esta librería):

```
template < typename Estado_t, typename Operaciones_t>
vector< typename Operaciones_t::value_type> profundidad_limitada(const Operaciones_t& operaciones,
    const Estado_t& inicial, const unsigned int& limite_profundidad, bool(*registrar_solucion) (
    const vector<typename Operaciones_t::value_type>&, const Estado_t& ) = NULL )
 {
    IA_REINICIAR_ESTADISTICAS;
    set<Estado_t> visitados;
    visitados.insert( inicial );
    ...
 }
```

con el mísmo código siguiendo el esquema usado en esta librería

```
template < typename Estado_t, typename Operaciones_t>
vector< typename Operaciones_t::value_type> profundidad_limitada(const Operaciones_t& operaciones
             , const Estado_t& inicial
             , const unsigned int& limite_profundidad
             , bool(*registrar_solucion)( const vector<typename Operaciones_t::value_type>&
                                     , const Estado_t& ) = NULL ) {
    IA_REINICIAR_ESTADISTICAS;
    set<Estado_t> visitados;
    visitados.insert( inicial );
...
 }
```

En el de más abajo resulta evidente que los parámetros de tipo const vector<>& y const Estado corresponden a la funcion recibida `registrar_solucion`, en el de más arriba darse cuenta de eso requiere un análisis de las lineas anteriores. El mismo caso se presenta en las sentencias for:

```
 for( typename Operaciones_t::const_iterator paso = solucion.begin(); paso != solucion.end();
      ++ paso)
     anterior = actual;
 actual = X;
```

¿ ++paso era parte del cuerpo del for?, aquí se ve mas claramente que ++paso no es el cuerpo del for, especialmente por el punto y coma:

```
 for( typename Operaciones_t::const_iterator paso = solucion.begin(); paso != solucion.end()
                                                                     ; ++ paso)
     anterior = actual;
 actual = X;
```