```cpp

void Graph::bfs( std::string start )
{

   // inicializamos TODOS los vértices a su valor por defecto (con respecto a
   // los atributos que tienen que ver con el algoritmo bfs)
    for( auto it = this->vertices.begin(); it != this->vertices.end(); ++it ){

        (it->second).set_color( Vertex::Colors::BLACK );
        // el vértice no ha sido descubierto ni visitado

        (it->second).set_distance( 0 );
        (it->second).set_predecesor( "Nil" );
    }


   get_vertex( start )->set_color( Vertex::Colors::GRAY );
   // marcamos al vértice de inicio como descubierto


#ifdef DEBUG
   std::cout << "Antes: \n";
    for( auto it = this->vertices.begin(); it != this->vertices.end(); ++it ){
      (it->second).print();
   }
#endif


   std::deque<std::string> queue;
   // una 'list' serviría al mismo propósito, pero 'deque' tiene más espíritu de
   // cola que una lista plana. 'deque' es por: double ended queue


   queue.push_back( start );
   // insertamos (encolamos) al vértice de inicio en la parte trasera de la cola

   while( not queue.empty() ){

#ifdef DEBUG
      print_queue( queue );
#endif

      std::string next_vertex = queue.front(); queue.pop_front();
      // obtenemos el NOMBRE asociado al vértice de trabajo ...
      // (desencolar toma dos operaciones: .front() y .pop_front())


#ifdef DEBUG
      std::cout << "Vértice de trabajo: " << next_vertex << std::endl;
#endif

      Vertex* vertex = get_vertex( next_vertex );
      // obtenemos una REFERENCIA al vértice de trabajo ...

      std::list<Vertex>* v = vertex->get_neighbors();
      // obtenemos el original de la LISTA de vértices vecinos ...

      for( auto w = v->begin(); w != v->end(); ++w ){

         Vertex* neighbor = get_vertex( w->get_name() );

         if( neighbor->get_color() == Vertex::Colors::BLACK ){

#ifdef DEBUG
            std::cout << "* procesando al vértice: " << neighbor->get_name() << std::endl;
#endif

```

```
68              neighbor->set_color( Vertex::Colors::GRAY );
69              // vertice descubierto ...
70
71              neighbor->set_distance( vertex->get_distance() + 1 );
72              // establecemos la distancia desde 'start' ...
73
74              neighbor->set_predecesor( vertex->get_name() );
75              // establecemos al predecesor ...
76
77              queue.push_back( neighbor->get_name() );
78              // encolamos al vértice recién descubierto
79           }
80   #ifdef DEBUG
81           else{
82              std::cout << "El vértice: " << neighbor->get_name();
83              std::cout << " ya había sido descubierto. No se hace nada\n";
84           }
85   #endif
86        }
87
88        vertex->set_color( Vertex::Colors::WHITE );
89        // vértice visitado
90     }
91
92   #ifdef DEBUG
93     std::cout << "Después: \n";
94      for( auto it = this->vertices.begin(); it != this->vertices.end(); ++it ){
95        (it->second).print();
96     }
97   #endif
98   }
```