In [2]:

```
import collections
import heapq
import types


class ListaNodos(collections.deque):
    def anadir(self, nodo):
        self.append(nodo)

    def vaciar(self):
        self.clear()

    def __contains__(self, nodo):
        return any(x.estado == nodo.estado
                   for x in self)


class PilaNodos(ListaNodos):
    def sacar(self):
        return self.pop()


class ColaNodos(ListaNodos):
    def sacar(self):
        return self.popleft()


class ColaNodosConPrioridad:
    def __init__(self):
        self.nodos = []
        self.nodo_generado = 0

    def anadir(self, nodo):
        heapq.heappush(self.nodos, (nodo.heuristica, self.nodo_generado, nodo))
        self.nodo_generado += 1

    def sacar(self):
        return heapq.heappop(self.nodos)[2]

    def vaciar(self):
        self.__init__()

    def __iter__(self):
        return iter(self.nodos)

    def __contains__(self, nodo):
        return any(x[2].estado == nodo.estado and
                   x[2].heuristica <= nodo.heuristica
                   for x in self.nodos)


class NodoSimple:
    def __init__(self, estado, padre=None, accion=None):
        self.estado = estado
        self.padre = padre
        self.accion = accion

    def es_raiz(self):
        return self.padre is None

    def sucesor(self, accion):
        Nodo = self.__class__
        return Nodo(accion.aplicar(self.estado), self, accion)
```

```
    def solucion(self):
        if self.es_raiz():
            acciones = []
        else:
            acciones = self.padre.solucion()
            acciones.append(self.accion.nombre)
        return acciones

    def __str__(self):
        return 'Estado: {}'.format(self.estado)


class NodoConProfundidad(NodoSimple):
    def __init__(self, estado, padre=None, accion=None):
        super().__init__(estado, padre, accion)
        if self.es_raiz():
            self.profundidad = 0
        else:
            self.profundidad = padre.profundidad + 1

    def __str__(self):
        return 'Estado: {0}; Prof: {1}'.format(self.estado, self.profundidad)


class NodoConHeuristica(NodoSimple):
    def __init__(self, estado, padre=None, accion=None):
        super().__init__(estado, padre, accion)
        if self.es_raiz():
            self.profundidad = 0
            self.coste = 0
        else:
            self.profundidad = padre.profundidad + 1
            self.coste = padre.coste + accion.coste_de_aplicar(padre.estado)
        self.heuristica = self.f(self)

    @staticmethod
    def f(nodo):
        return 0

    def __str__(self):
        return 'Estado: {0}; Prof: {1}; Valoración: {2}; Coste: {3}'.format(
            self.estado, self.profundidad, self.heuristica, self.coste)


class BusquedaGeneral:
    def __init__(self, detallado=False):
        self.detallado = detallado
        if self.detallado:
            self.Nodo = NodoConProfundidad
        else:
            self.Nodo = NodoSimple
        self.explorados = ListaNodos()
        self.num_explorados = 0

    def es_expandible(self, nodo):
        return True

    def expandir_nodo(self, nodo, problema):
        return (nodo.sucesor(accion)
                for accion in problema.acciones_aplicables(nodo.estado))

    def es_nuevo(self, nodo):
        return (nodo not in self.frontera and
```

```
                nodo not in self.explorados)

    def buscar(self, problema):
        self.frontera.vaciar()
        self.explorados.vaciar()
        self.frontera.anadir(self.Nodo(problema.estado_inicial))
        self.num_explorados = 0
        while True:
            if not self.frontera:
                return None
            nodo = self.frontera.sacar()
            self.num_explorados += 1
            if self.detallado:
                print('{0}Nodo({1}): {2}'.format('  ' * nodo.profundidad, self.num_e
                # print('{0}Nodo: {1}; Frontera: {2}'.format('  ' * nodo.profundidad
            if problema.es_estado_final(nodo.estado):
                return nodo.solucion()
            self.explorados.anadir(nodo)
            if self.es_expandible(nodo):
                nodos_hijos = self.expandir_nodo(nodo, problema)
                for nodo_hijo in nodos_hijos:
                    if self.es_nuevo(nodo_hijo):
                        self.frontera.anadir(nodo_hijo)


class BusquedaEnAnchura(BusquedaGeneral):
    def __init__(self, detallado=False):
        super().__init__(detallado)
        self.frontera = ColaNodos()


class BusquedaEnProfundidad(BusquedaGeneral):
    def __init__(self, detallado=False):
        super().__init__(detallado)
        self.frontera = PilaNodos()
        self.explorados = PilaNodos()

        def anadir_vaciando_rama(self, nodo):
            if self:
                while True:
                    ultimo_nodo = self.pop()
                    if ultimo_nodo == nodo.padre:
                        self.append(ultimo_nodo)
                        break
            self.append(nodo)
        self.explorados.anadir = types.MethodType(anadir_vaciando_rama,
                                                  self.explorados)


class BusquedaEnProfundidadAcotada(BusquedaEnProfundidad):
    def __init__(self, cota, detallado=False):
        super().__init__(detallado)
        self.Nodo = NodoConProfundidad
        self.cota = cota

    def es_expandible(self, nodo):
        return nodo.profundidad < self.cota


class BusquedaEnProfundidadIterativa:
    def __init__(self, cota_final, cota_inicial=0, detallado=False):
        self.cota_inicial = cota_inicial
        self.cota_final = cota_final
        self.detallado = detallado
```

```
    def buscar(self, problema):
        for cota in range(self.cota_inicial, self.cota_final):
            bpa = BusquedaEnProfundidadAcotada(cota, self.detallado)
            solucion = bpa.buscar(problema)
            if solucion:
                return solucion


class BusquedaOptima(BusquedaGeneral):
    def __init__(self, detallado=False):
        super().__init__(detallado)
        self.Nodo = NodoConHeuristica
        self.Nodo.f = staticmethod(lambda nodo: nodo.coste)
        self.frontera = ColaNodosConPrioridad()
        self.explorados = ListaNodos()
        self.explorados.__contains__ = types.MethodType(
            lambda self, nodo: any(x.estado == nodo.estado and
                                   x.heuristica <= nodo.heuristica
                                   for x in self),
            self.explorados)


class BusquedaPrimeroElMejor(BusquedaOptima):
    def __init__(self, h, detallado=False):
        super().__init__(detallado)
        self.Nodo.f = staticmethod(lambda nodo: h(nodo.estado))


class BusquedaAEstrella(BusquedaOptima):
    def __init__(self, h, detallado=False):
        super().__init__(detallado)
        self.Nodo.f = staticmethod(lambda nodo: nodo.coste + h(nodo.estado))
```