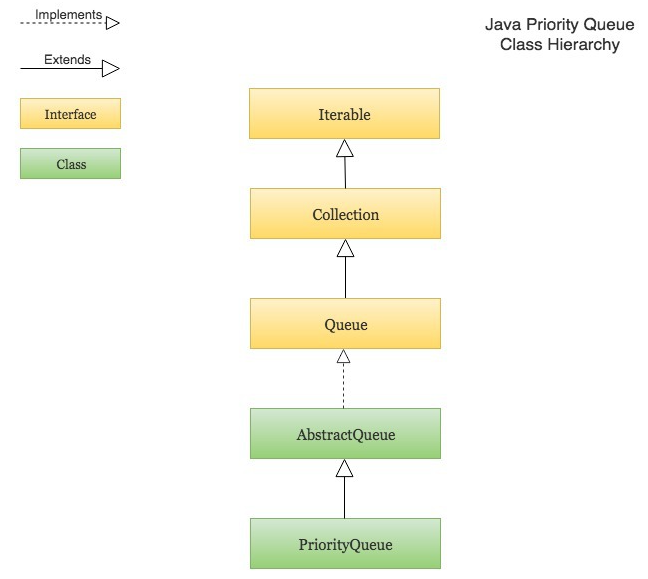
**Java Tips**

* LinkedList:
  1. Part of the *Collection* framework in *java.util.package*
  2. Linear data structure – elements are not stored in contiguous locations
  3. Every element is a separate object with a data part and address part.
  4. The elements are linked using pointers and addresses.
  5. Suitable for dynamic manipulation, such as insertion/deletion
  6. **Limitation**: elements cannot be accessed directly. Instead, we need to start from the head and follow through the link to reach the node we wish to access.
* PriorityQueue:
  1. Special type of queue wherein all the elements are **ordered** according to their natural ordering or based on a custom *Comparator* supplied at the time of creation.
  2. The **front**: least element; the **rear**: the greatest element.



|  |
| --- |
|  |
|  |

**Search Algorithm**

|  |  |  |
| --- | --- | --- |
|  | **Advantage** | **Disadvantage** |
| **DFS** | * Require less space: O(d) | * Delay search for node close to the root but not belonging to the first few deep substrees |
| **BFS** | * Good for local search | * Require more space: O(n) |

1. Iterative deepening depth-first search (IDDFS) is invented to inherit the advantages of both DFS and BFS.
   1. The algorithm visits the search tree in the same order as DFS
   2. The cumulative order in which nodes are first visited is BFS
2. A\* Search
   1. Find path that minimizes *f(x)*:
   2. *F(x) = g(x) + h(x)*
   3. *G(x)* is the cost function, *h(x)* is the estimated cost from the current to destination nodes.

**Heuristics vs meta-heuristics**

1. Both heuristics and meta-heuristics aim to get reasonably good optimal solution (best possible solution is not guaranteed) in a short time.
2. However, heuristics is specific to the underlying problem at hand, whereas meta-heuristics are problem independent and therefore could be applied to any kinds of problems.

**Tabu Search**

1. Local search algorithm such as hill climbing is gradient based approach. The algorithm is easily stuck in local optimum and it takes repetitive initializations of the algorithm to detect the global maximum.
2. Tabu Search is designed to help the optimum-searching algorithm escape from the local optimum by prohibiting returning to the previously visited state.
3. When a move is prohibited, aka made tabu, it is added to a list called **tabu list**, usually implemented as a queue. New item inserted into the list could be store for as many iterations as the size of the queue, usually referred to as **the tabu tenure**.
4. **Aspiration** criteria - Sometimes, we do allow tabu moves for example, when the given move allows a new global best solution.