Trait BnbAwareContainer


```
pub trait BnbAwareContainer<S: Subproblem> {
    // Required methods
    fn push_with_incumbent(&mut self, item: S, score: Option<&S::Score>);

    fn pop_with_incumbent(&mut self, score: Option<&S::Score>) -> Option<S>;
}
```

A container for subproblem objects, which is used to store unvisited nodes of the subproblem tree.

A container provides an interface to push and pop items and:

1. Defines order in which elements will be popped;
2. May implement additional features, such as early stopping, deciding not to push/return some elements based on the value of the incumbent, etc.

Required Methods

`fn push_with_incumbent(&mut self, item: S, score: Option<&S::Score>)`

Add `item` to the container.

`score` is the objective score of the current incumbent (if any). The container may decide not to add an item if it's known to be worse than the incumbent ("eager" evaluation strategy).

`fn pop_with_incumbent(&mut self, score: Option<&S::Score>) -> Option<S>`

Get an item from the container. `score` is the objective score of the current incumbent (if any). The container may decide to skip items that are known to be worse than the incumbent ("lazy" evaluation strategy).

Returns `None` iff the container is exhausted (i.e., there's no more feasible subproblems to process).

After `.pop_with_incumbent` returns `None`, the object should not be used anymore: calling either `.push_with_incumbent` or `.pop_with_incumbent` will have unspecified results.

Implementations on Foreign Types

`impl<S: Subproblem> BnbAwareContainer<S> for VecDeque<S>`

This implementation for `VecDeque` is an implementation of the extra-eager strategy: it checks against the ...

`impl<S: Subproblem> BnbAwareContainer<S> for Vec<S>`

This implementation for `Vec` is an implementation of the extra-eager strategy: it checks against the incum …

Implementors