CO202: Coursework 1

Group #number

Autumn Term, 2019

The source of this document is `Submision.lhs`, and should form the basis of your report as well as contain all the code for your submission. You should remove text (such as all the text in this section) that is here for your information only and that does not contribute to your submission. You should start by modifying the `\author{}` command above to include your group number.

The source code of the provided `Submission.lhs` contains code and comments that are hidden from the final `pdf` file, so you should inspect it carefully. For instance, the code declares the use of various language features that are used in this code base. You can learn more about these language features in the language extensions section of the GHC documentation at `https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/glasgow_exts.html` if you wish, but for the most part you need not worry about them.

The following imports various modules that are used. You should avoid depending on any libraries other than those distributed with GHC: `base` and `containers` ought to contain everything you need.

All of the necessary types and definitions from the specification of this coursework have been given to you in the source of this document. You need not repeat that code in your submission, but it is required within the `\begin{code}` and `\end{code}` markers so that it can be compiled.

Before submitting your coursework, you should ensure that your code compiles properly. Use the following command with the supplied `Submission.lhs-boot` file to check that it can be marked:

```
ghc -fforce-recomp -c Submission.lhs-boot Submission.lhs
```

This checks to see if all the type signatures of exposed functions are as expected.

Make sure that the problems you are solving are clearly indicated. Using a section is a good idea. You should endeavor to concisely explain the code you have written. Feel free to make use of your own margin notes, and do please remove this one.

Problem 1: Dynamic Knapsack

```
knapsack' :: forall name weight value .
  (Ix weight, Num weight, Ord value, Num value) =>
  [(name, weight, value)] -> weight -> value
knapsack' wvs c = table ! c
  where
    table :: Array weight value
    table = tabulate (0,c) mknapsack
```

```
    mknapsack :: weight -> value
    mknapsack c = undefined
```

Problem 2: Knapsack Elements

```
knapsack'' :: forall name weight value .
  (Ix weight, Num weight, Ord value, Num value) =>
  [(name, weight, value)] -> weight -> (value, [name])
knapsack'' wvs c = table ! c
  where
    table :: Array weight (value, [name])
    table = tabulate (0,c) mknapsack

    mknapsack :: weight -> (value, [name])
    mknapsack c = undefined
```

Problem 3: Bounded Knapsack

```
bknapsack
  :: (Ord weight, Num weight, Ord value, Num value)
  => [(name, weight, value)] -> weight -> (value, [name])
bknapsack = undefined
```

Problem 4: Reasonable Indexes

Problem 5: Bounded Knapsack Revisited

```
bknapsack' :: forall name weight value .
  (Ord weight, Num weight, Ord value, Num value) =>
  [(name, weight, value)] -> Int ->
  weight -> (value, [name])
bknapsack' = undefined
```

Problem 6: Dynamic Bounded Knapsack

```
bknapsack'' :: forall name weight value .
  (Ord name, Ix weight, Ord weight, Num weight,
    Ord value, Num value) =>
  [(name, weight, value)] -> weight -> (value, [name])
bknapsack'' = undefined
```

Problem 7: Dijkstra Dualized

Problem 8: Heap Operations

```
data Heap a = Heap (a -> a -> Ordering) (Tree a)
data Tree a = Nil | Node Int (Tree a) a (Tree a)

instance PQueue Heap where
  toPQueue = undefined
  fromPQueue = undefined

  priority :: Heap a -> (a -> a -> Ordering)
  priority = undefined

  empty :: (a -> a -> Ordering) -> Heap a
  empty p = undefined

  isEmpty :: Heap a -> Bool
  isEmpty = undefined

  insert :: a -> Heap a -> Heap a
  insert = undefined

  delete :: a -> Heap a -> Heap a
  delete = undefined

  extract :: Heap a -> a
  extract = undefined

  discard :: Heap a -> Heap a
  discard = undefined

  detach :: Heap a -> (a, Heap a)
  detach = undefined
```

Problem 9: Adjacency List Graphs

```
newtype AdjList e v = AdjList [(v, [e])]

instance (Eq e, Edge e v) => Graph (AdjList e v) e v where
  vertices (AdjList ves)    = undefined
  edges (AdjList ves)       = undefined
  edgesFrom (AdjList ves) s = undefined
  edgesTo   (AdjList ves) t = undefined
  velem v (AdjList ves)     = undefined
  eelem e (AdjList ves)     = undefined
```

Problem 10: Conflict Zones

```
conflictZones :: GameState -> PlanetId -> PlanetId
  -> ([PlanetId], [PlanetId], [PlanetId])
conflictZones g p q = undefined
```