キャッシュ（宿題４）

横山奈穂

概要

このプログラムは、キャッシュを管理するデータ構造を実装している。キャッシュは Least Recently Used（LRU）アルゴリズムを用いて、最近アクセスされた N 個のページを保存し、古いページはキャッシュから削除する。ハッシュテーブルと配列を組み合わせることで、検索、挿入、および削除操作をほぼ O(1)で実行する。

主な関数

1. ページにアクセスしてキャッシュする（access_page）
   引数: url, contents（任意の型）
   戻り値: なし

2. キャッシュしたページの一覧を取得する（get_page）
   引数: なし
   戻り値: キャッシュしたページの配列（アクセス順にソート済み）

アルゴリズム

1. キャッシュ方法
   データの追加は、そのデータが既にハッシュテーブル内に存在するか否かで場合分けされ、以下のように行われる。どちらも、クラス変数である index（最も古いデータの場所を示すインデックス、新しくアクセスしたページと入れ替えたい場所を示す）を使用する。

   1.1 データがハッシュテーブル内に存在する場合
   以下のようにして、アクセス順配列を更新する。

   1. Index の場所にあるデータを別の変数に保存し、アクセスしたページと入れ替える
   2. 配列をたどって、アクセスしたページにたどり着くまで保存した変数とデータを入れ替える

   1.2 データがハッシュテーブル内に存在しない場合
   以下のようにして、アクセス順配列及びキャッシュテーブルを更新する。

   1. 配列の index 番のデータを取得する

2．取得したデータに対応するキーをハッシュテーブルから削除する（削除の方法は宿題 1 のハッシュテーブルドキュメントを参照）

3．配列の index 番に新しいデータを入れる

4．新しいデータをハッシュテーブルに追加する（追加の方法は宿題 1 のハッシュテーブルドキュメントを参照）

2．キャッシュしたページ一覧の取得方法

アクセス順配列がページの url 一覧を持っているので、それを index を基準にさかのぼる形で返す。ただし、N 回分のデータの追加が行われる前だと None というデータが入ってしまうため、これを返す前に取り除く処理にした。

工夫した点

1. index の更新方法

キャッシュの大きさが固定長（最大のサイズが N で固定）な事を利用し、index を 1 ずつ増加させて N で割った余りをとる方法で更新した。これにより、配列の端までいったら逆の端に移動するような更新となり、場合分けの処理を書かなくても index を更新する事が出来る。