コンピュータサイエンス実験第二 B(担当：古賀)

ハッシュを使った類似検索

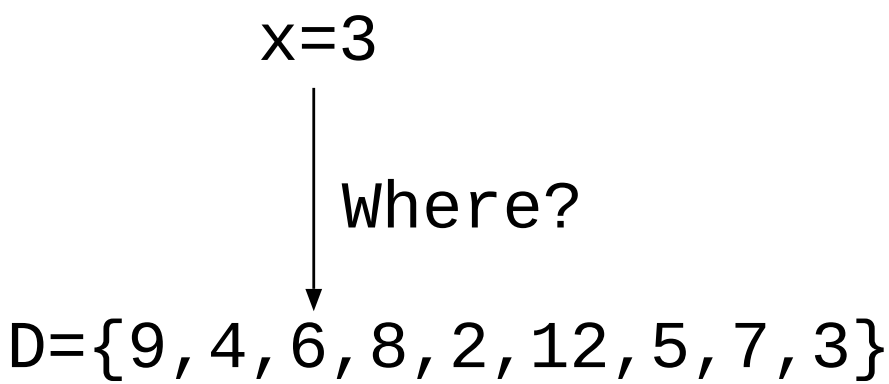
プロジェクト概要

このプロジェクトでは類似検索を高速に実現する技術について学ぶ。とくに、集合の類似検索をハッシュで実現する技術である Min-Hash を C++で実装する。また、デバッガなどプログラミングに役立つツールの習得もプロジェクトの目的である。

データ探索

探索とはデータ群 D から「条件を満たすデータを探す」こと。
データ構造とアルゴリズムの授業では

- D: 数字の集合
- 探索対象: 数字 x を見つける。例 :「学生証番号をキーとして検索」



データ探索の高速処理

データ探索は x を D の全要素と比較すれば自明に解ける。しかしそれでは、実効速度が遅い。比較を省略することで、データ探索を高速に実現できる。

1. 2 分探索木
2. ハッシュ

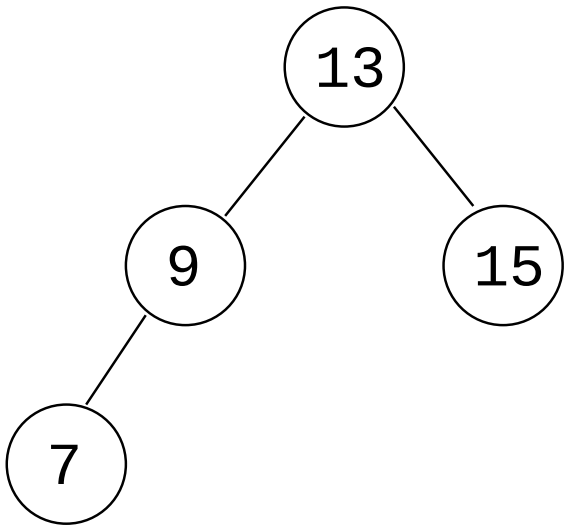


図: 2 分探索木

ハッシュ

元々はクエリと同一データを高速検索するための技術である。
ハッシュテーブルを使って探索すべきデータを絞り込んで高速化を実現する。h をハッシュ関数とする。

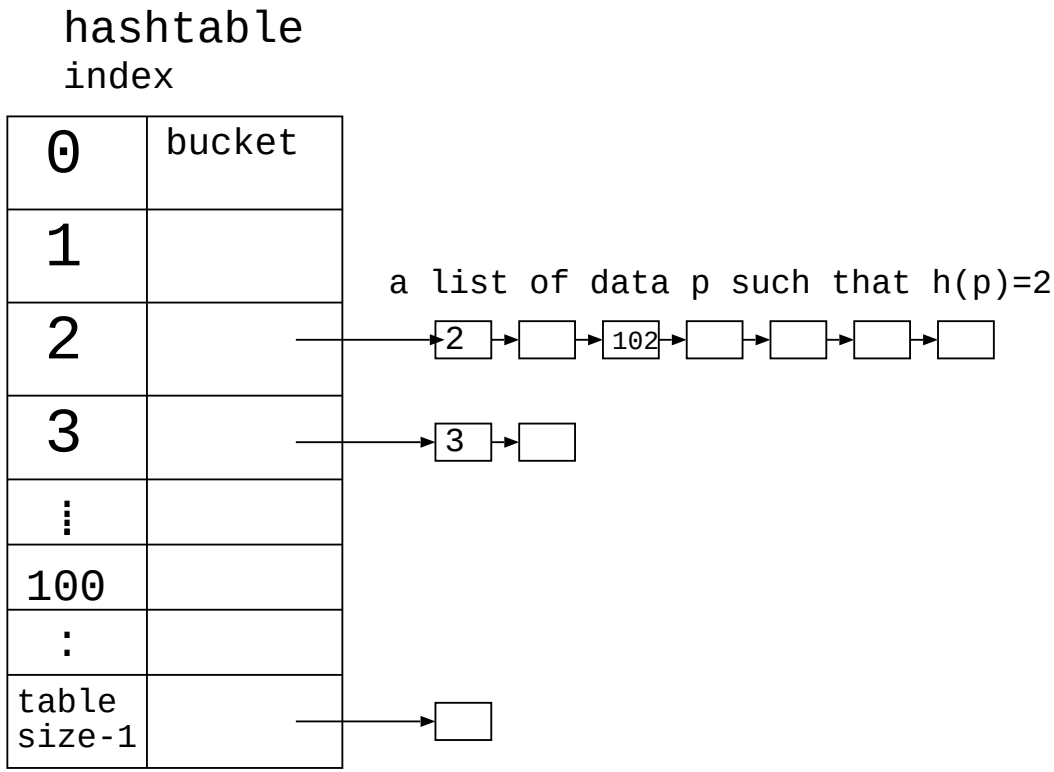


図: チェイン法でのハッシュテーブル

ハッシュテーブルへの登録

$\forall p \in D$ についてハッシュ値 $h(p)$ を計算し、$h(p)$ の値をインデックスとしてハッシュテーブルに登録。ハッシュテーブルの各エントリーをバケツと呼ぶ。インデックス値が i であるバケツを $b(i)$ と記述する。
データベース内の異なるデータが同一ハッシュ値になることをハッシュ値の衝突 (collision) と呼ぶ。collision の対処方法は 2 つある。

- チェイン法
- 開番地法 (Open Addressing)

チェイン法

各バケツは、リストを使ってデータを複数保持。データ p は必ずバケツ $b(h(p))$ に格納される。逆にバケツ $b(i)$ はハッシュ値 $h(p) = i$ となるデータをすべて保持。

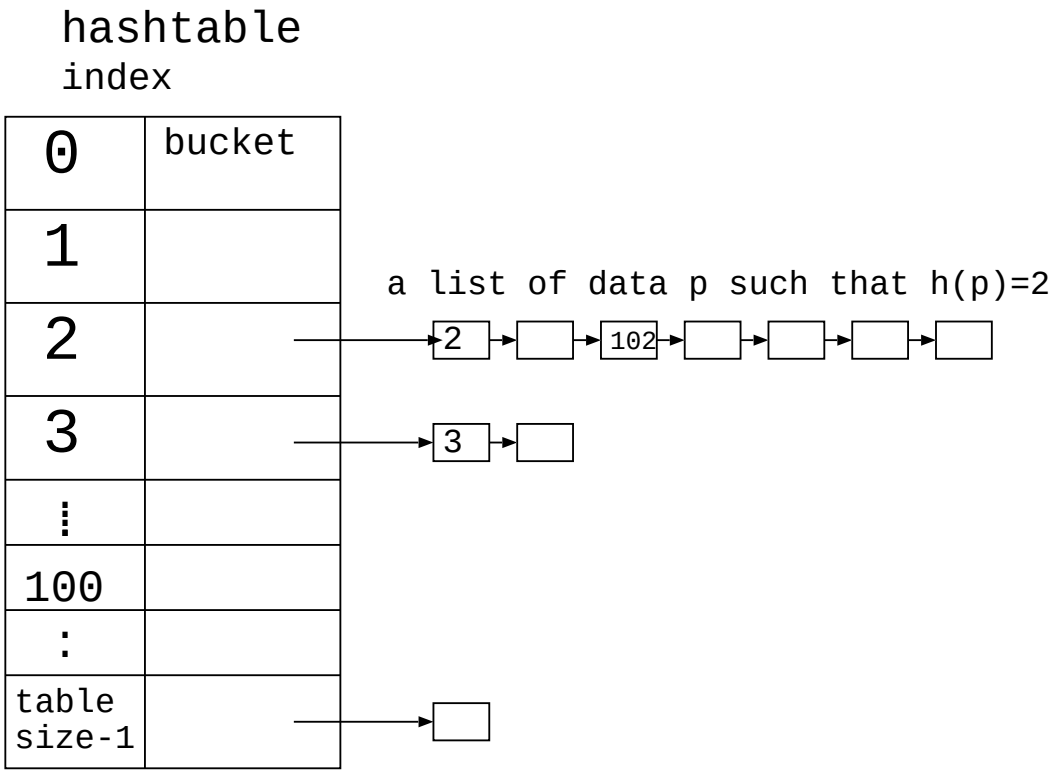


図: チェイン法でのハッシュテーブル

開番地法 (open addressing)

各バケツはデータを 1 つだけ持つ。点 p を登録しようとする時、

- if $b(h(p))$ が空 $\rightarrow$ p をバケツ $b(h(p))$ に登録。
- else p を $b(h(p))$ より後方の最初の空バケツに登録。

よって、p はバケツ $b(h(p))$ に格納されるとは限らない。

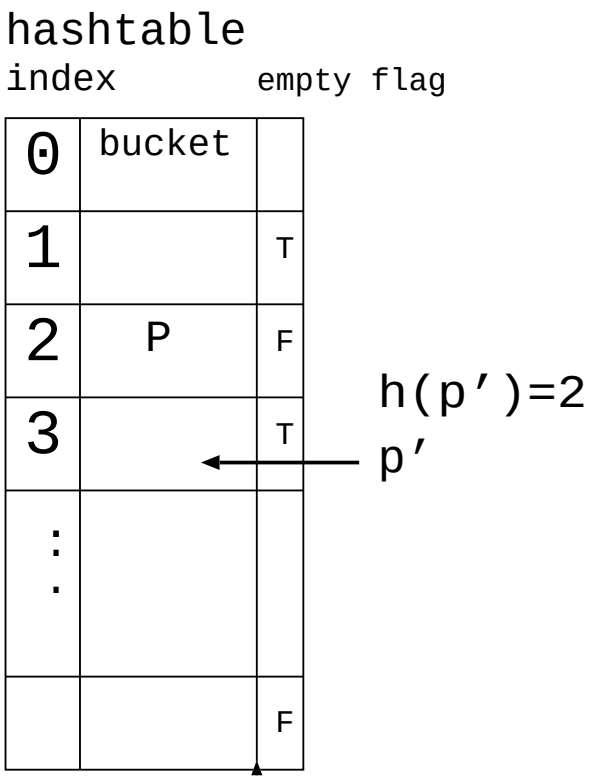


図: 開番地法でのハッシュテーブル

データ x の探索

1. x のハッシュ値 $h(x)$ を計算。
2. ハッシュテーブルにおいてバケツ $b(h(x))$ を調べ、x が存在するかを確認。
   - チェイン法では $b(h(x))$ のリストを調べる。アクセス性能はバケツ $b(h(x))$ 内の点数で決まる。
   - 開番地法では、バケツ $b(h(x))$ からスタートして、x を格納するバケツを探索。
     - バケツが空でない場合、x を格納しているかを確認。x 以外を格納している場合は次のバケツに進む。
     - 空バケツ出現 $\rightarrow$ 検索失敗

   探索性能はハッシュテーブルでバケツを走査する数で決まる。

チェイン法と開番地法のどちらも、$h(x)$ をうまく設計すれば、実用上 $O(1)$ のアクセス時間を達成。

ハッシュテーブルのデータ構造

ハッシュテーブルはバケツを table_size 個並べたもの。

```
typedef unsigned int uint;

typedef struct HashTable {
    uint table_size; // ハッシュテーブルサイズ
    vector<Bucket> buckets; //バケツの動的配列
} HashTable;
```

- vector は C++における動的配列。最初に要素数を指定しないことが可能。

チェイン法でのハッシュバケツ

```
typedef uint Item;

typedef struct Bucket{
     vector<Item> Items; //リスト構造
} Bucket;
```

開番地法でのハッシュバケツ

```
typedef struct Bucket{
  bool isempty //バケツが空かどうかを判定する論理型変数，初期値はtrue
  Item ID; //バケツが格納するデータ
} Bucket;
```

類似検索

データ集合 S から, 問い合わせデータ q に近いデータを探す操作を**類似検索**と呼ぶ。q はクエリ (query) と呼ばれる。$\mathrm{sim}(p, q)$ は p,q 間の類似度を表す。

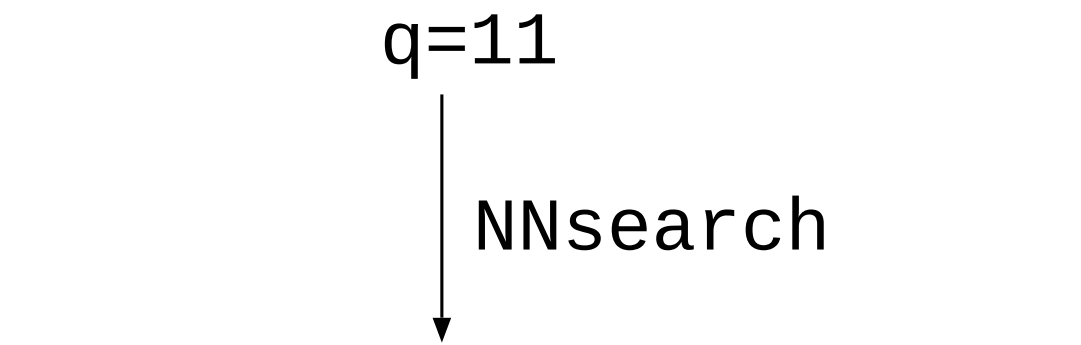
1. NN 探索 (nearest neighbor search): q に最も近い S のデータを探す。
2. k-NN 探索 (k nearest neighbor search): q に最も近いデータを S から k 個探す。
3. レンジ探索 (range search): 領域 (range) 内にあるデータを見つける問題をレンジ探索という。類似検索に関する場合は q から類似度 r 以上の S のデータを探す問題。

$$\text{return } p \in S \text{ such that } \mathrm{sim}(p, q) \geq r.$$

実験では NN 探索と k-NN 探索を取り扱う。NN 探索は、q と S の全データとの距離を計算すれば自明にできる。

いかに距離計算の回数を減らせるかが NN-探索では重要

NN 探索の例



Answer=12

集合間類似検索

集合を対象とした類似検索。

$$Q = \{\text{みかん、りんご、メロン、グレープフルーツ}\}$$

- NN 探索: クエリ集合 Q と最も類似した集合を集合のデータベース S から探す

類似検索の結果は，2 集合間の類似度 $\mathrm{sim}(p, q)$ をどう定めるかで変化する。

- C_i, C_j: 集合
- $\mathrm{sim}(C_i, C_j)$: 集合に対する類似度

集合間類似度

代表的な集合間に対する類似度としては Jaccard 係数や Dice 係数が知られている。今回の実験では、主に Jaccard 係数を用いる。

- Jaccard 係数：直観的には 2 集合の共通要素の割合を表し、値域は 0 から 1 の間。

$$\mathrm{sim}(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}. \tag{1}$$

- Dice 係数: やはり、値域は 0 から 1 の間。

$$\mathrm{sim}(C_i, C_j) = \frac{2|C_i \cap C_j|}{|C_i| + |C_j|}. \tag{2}$$

- 積集合サイズ:　共通要素数

$$|C_i \cap C_j| \tag{3}$$

Jaccard 係数

$$
\begin{aligned}
C_1 &= \{\text{みかん、りんご、メロン、グレープフルーツ}\} \\
C_2 &= \{\text{みかん、いちご、梨、ぶどう}\} \\
C_3 &= \{\text{みかん、いちご、梨、桃}\}
\end{aligned}
$$

3 集合の Jaccard 係数は、

$$
\begin{aligned}
\mathrm{sim}(C_1, C_2) &= \frac{1}{7}. \\
\mathrm{sim}(C_2, C_3) &= \frac{3}{5}. \\
\mathrm{sim}(C_3, C_1) &= \frac{1}{7}.
\end{aligned}
$$

集合間類似検索の応用

文書間の類似性判定やマーケット分析など様々な分野で使われる。

- **マーケット分析**
  商品 C_i に対して、「C_i を購入した客」の集合を考える。C_i と C_j の Jaccard 係数大 $\rightarrow$ C_i と C_j は同時に買われる商品
- **文書間の類似性判定**
  文書 C_i に対して、「C_i に含まれる単語」の集合を考える。

実験に使うデータ

手書き数字データベース MNIST を使用する。
http://yann.lecun.com/exdb/mnist/
学習画像 60000 枚、テスト画像 10000 枚。画像の大きさは 28x28=784 画素。
各テスト画像に対し、最も類似した学習画像を学習画像データベースから探す。

- 画像の集合表現　グレースケール画像を黒でない画素の位置集合で表現。
  - 集合の要素は 0 から 783 までの整数

開発手順

1. MNIST のデータをプログラムから読み込めるようにする。
2. クエリ q と全要素を比較する単純な類似検索法 (Brute Force) の実装。
3. Min-Hash の実装
4. Min-Hash 及び Brute Force を MNIST データセットに適用し、実行速度を比較することにより Min-Hash の高速性を確認する。

課題 1,2

1. チェイン法のサンプルプログラムを開番地法に書き直せ。
2. MNIST のデータ test-images.txt の各行を読み込んで、集合として表現するプログラムを作れ。グレースケール画像を黒 (画素値 0) でない画素の位置集合により集合表現する。集合の型は vector<Item> とし、Item は昇順で並べること。
   mnistview.cc が非常に参考になるのでよく見ること。
   Item 型の 2 次元配列の宣言は以下のようになる。

   `typedef vector< vector<Item> > Database; //2次元配列の型定義`

   `Database db;`

   しかし，これだけでは db の 1 次元目の要素数 (vector<Item> が何個あるか) がわからないので困る．どこかで

   `db.resize(dbsize);`

   として vector<Item> が dbsize 個ある事を指定.

課題 3

2 つの集合間 C_1, C_2 で Jaccard 係数を計算する関数を作り、読み込んだすべての集合ペア間で Jaccard 係数を計算しなさい。

- 計算量は $O(|C_1| + |C_2|)$ となることを目指して下さい。
  ヒント：集合が要素の昇順にソートされていることを利用する。

```
//関数の宣言
double calc_jaccard(vector<Item> &s1, vector<Item> &s2){
}

//呼び出し側
double sim;
sim = calc_jaccrd(db[i],db[j]);
```

プログラムのコンパイル

gnu の C++コンパイラである g++を利用する。デバッグ時とリリース時でコンパイラオプションを変える習慣を身に付けよう。

- デバッグ時：　g++ -Wall -g
  -Wall は Warning レベルを上げる。このオプションで Warning が出ないプログラムが理想。
  -g はデバッガ gdb(gnu debugger) を利用するのに必要。デバグ情報を埋め込むため、実行速度は遅くなる。
- リリース時：　g++ -Wall -O2
  -O は最適化オプション。O の後ろの数字で最適化のレベルを指定。数字が大きいほど、最適化される。

必要なC++の知識(1)

標準テンプレート vector: C++の動的配列クラス。

- 配列のサイズを事前に指定せずに使える。
- 配列のサイズを事前に指定することも可能。
- 代表的なメンバ関数
  1. push_back(): 配列にデータを追加
  2. size(): 配列の要素数を返す
  3. clear(): 配列の全要素を消去

必要なC++の知識 (2)

参照： vectorを関数に渡す時に重宝される機能。
C++ではCと違い配列を値渡しができる。これによりアロー演算子が不要になる．しかし、値渡しではコピーが発生するため、配列サイズが大きいと大変遅くなる．
参照渡しは変数そのものを指定するので、コピーなしで渡せる。
ポインタ渡しと違って、アロー演算子も不要

ポインタ渡しと参照渡し

ポインタ渡しの例

```
void insert(Item data, HashTable *table){
  uint index = myhash(data, table->table_size);
  table->buckets[index].Items.push_back(data);
}

//呼び出し側
insert(db[i],&tab);
```

参照渡しの例

```
void insert(Item data, HashTable &table){
  uint index = myhash(data, table.table_size);
  table.buckets[index].Items.push_back(data);
}
//呼び出し側
insert(db[i],tab);
```

知っていると便利なC++の機能

- string 型： 文字列型
- stringstream 文字列ストリーム
- 入出力ストリーム cout, cin: C++での標準入出力、
- ファイルストリーム ifstream, ofstream： C++でのファイル入出力

サンプルコード

サンプルコード： google classroom で sample.zip という名前で配布。
train-images.txt は/home3/staff/ka109040/MICS/ というディレクトリよりローカルにコピー

chainhash.cc: チェイン法のサンプルプログラム

- ./chainhash テーブルサイズ　登録データ数
  1. ハッシュテーブルにデータを登録。
  2. 登録したデータからランダムに１つ選んで検索
- ハッシュ関数 myhash: テーブルサイズで割った余りを返す
- 登録データはプログラム内で合成

サンプルコード mnistview.cc

test-images.txt の中身を画像としてウィンドウ表示する

- 起動方法:
  ./mnistview (学習またはテスト) テキストデータファイル名
- テキストデータファイル
  - test-images.txt: MNIST のテスト画像をテキスト表現したもの。1 行が 1 画像に対応する。10000 行
  - train-images.txt: MNIST の学習画像をテキスト表現したもの。60000 行