便利なライブラリとその利用

解答例@github

便利なライブラリ(C++)

- C++ は各種データ構造を標準ライブラリに搭載 (C++ reference)
  - ソート(要素を順番に並べなおす)や配列の検索などのアルゴリズム系
  - Vector や map (連想配列:後述)などのコンテナ
  - 他にも、String や pair (二つの値の組)、complex (数字のペア、座標計算や複素数にも利用可)などいろいろ
- いくつか使ってみましょう。

便利なライブラリ（Java）

- Java は各種データ構造を標準ライブラリに搭載（参考）
  - リスト、集合やMap 索などの Collections Framework など
    - 比較・ソート関係も充実
  - 他にも、String や Point, Point2D（整数座標と浮動小数点座標）など
- いくつか使ってみましょう。

便利なライブラリ（python)

- Python は各種データ構造を組み込み型やライブラリとして搭載 (python3.10の標準ライブラリ)
  - 集合系
    - List（同種のデータの集まり、ソート可）
    - Tuple（異種データの集まり）
    - Dict（マッピング型、キーから対応するオブジェクト（値）へのマッピング）
  - String や heapq（ヒープキュー）、complex（座標計算や複素数にも利用可）などいろいろ

- いくつか使ってみましょう。

問題：2番目に大きな数を返す (C++)

- 5つの数字が与えられたとして、2番目に大きな数字を返す
- 解1：一番大きなものを探してから、それを省いてから、また一番大きなものを選んでみよう。
- 解2：面倒だから、ソートして2番目の要素をとればいいじゃん。
  - sort 関数
    - vec.begin(), end() は、最初と最後を表すiterator と呼ばれる物

解答例@github

```
int main(void) {
  const int n = 5;
  vector<int> vec(n);
  for (int i = 0; i < n; i++) {
    cin >> vec[i];
  }
  sort(vec.begin(), vec.end()); // 小さい順
  cout << vec[n-2] << endl;
}
```

問題：2番目に大きな数を返す (Java)

- 5つの数字が与えられたとして、2番目に大きな数字を返す
- 解１：一番大きなものを探してから、それを省いてから、また一番大きなものを選んでみよう。
- 解２：面倒だから、ソートして2番目の要素をとればいいじゃん
  - 配列のソートなどは Arraysクラスで対応可
  - List クラス（ArrayList）などにデータ格納してもよい
    - ➢Collections でソート

解答例

```
Scanner input = new Scanner(System.in);
int n = 5;
int[] numbers = new int[5];
for (int i = 0; i < 5; i++) numbers[i] = input.nextInt();
Arrays.sort(numbers);
System.out.println(numbers[n - 2]);
```

```
ArrayList<Integer> list = new ArrayList<>();
for (int i = 0; i < 5; i++) list.add(input.nextInt());
Collections.sort(list);
System.out.println(list.get(n-2));
```

ArrayList 版

問題：2番目に大きな数を返す (python)

- 5つの数字が与えられたとして、2番目に大きな数字を返す
- 解１：一番大きなものを探してから、それを省いてから、また一番大きなものを選んでみよう。
- 解２：面倒だから、ソートして2番目の要素をとればいいじゃん。
  - list のsort メソッド
    - ➢list の順番を変えて小さい順に
  - sorted(list) 関数
    - ➢list の順番は変更せず、小さい順のリストを新規作成

解答例@github

```
def main():
    n = 5
    numbers = [int(input()) for _ in range(n)]
    numbers.sort()
    print(numbers[n - 2])
    # あるいは、numbers をソートした結果を新規作成する場合
    # print(sorted(numbers)[n-2])
```

問題AC@レギオ神戸問題セット

問題文@github

問題：発注集計

- 商品名と個数のペアを複数受け取り、商品ごとに個数を返す
  - 商品の表示順は、短い名前優先で、同じ長さならアルファベット順

- 今回利用するデータ構造: map（連想配列）
  - key と value のペアを保持
  - key には、int や string といった順序付きの値を用いる
  - 格納された key-value ペアを、順番にアクセスできる

- Java: TreeMap が相当
- python: dict クラスと sort関数を使うのが便利

```
map<string, int> orders;
  { “A” : 20,
    “B” : 20,
    “AB” : 20 }
```

問題：発注集計（続き，C++）

- プログラム例
  - string name, int num を取り込みマップに登録していく。
  - orders[name] = ...; で登録可能
  - 取得は orders[name] でOK.
    対応する key が未登録の場合は 0 (default 値)が返される。
  - Key が登録されているか確認したい場合は、orders.count(name) が 0 か 1 か確認すれば OK.

program@github

```
int n;
cin >> n;
map<string, int> orders;
for (int i = 0; i < n; i++) {
  string name;
  int num;
  cin >> name;
  cin >> num;
  int prev = orders[name];
  orders[name] = prev + num;
}
```

問題：発注集計（続き2，C++）

- 今回は、商品名の長さが短い順に出力せよという変な問題。
- map は並び順も登録できる（使い方を覚えるのがちょい面倒かも）

```
// stringの比較器
auto cmp = [](string a, string b) {
  // string の長さが違うときは、長さで判断
  if (a.length() != b.length()) return a.length() < b.length();
  // 長さが同じ時は、辞書順で比較
  return a < b;
};
// 比較器 cmp を指定した map の作成
map<string, int, decltype(cmp)> orders(cmp);
```

問題：発注集計（続き，python）

- プログラム例
  - string name, int num を取り込み dict に登録していく。
  - orders[name] = ...; で登録可能
  - 取得は orders[name] でもOK だが、要素があるか事前チェック（key in orders で判定）が必要
  - 右では、orders.get(name, 0) で要素が含まれない場合は default 値 0 が返されるようにしている

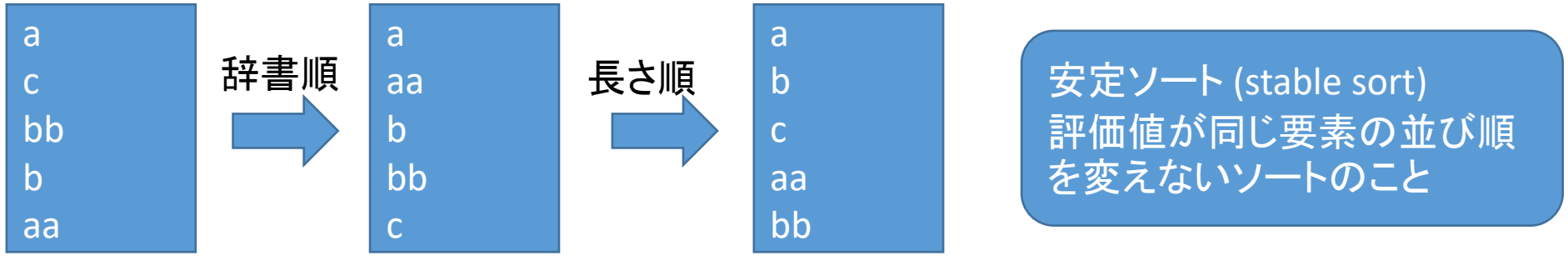
program@github

```
n = int(input())
orders = {}
for _ in range(n):
    name, num0 = input().split()
    prev = orders.get(name, 0)
    orders[name] = prev+int(num0)
```

問題：発注集計（続き2，python）

- 今回は、商品名の長さが短い順に出力せよという変な問題。
- dict は並び順指定できないので、key を2回ソートして対応

```
sorted_keys = sorted(orders.keys())  # key の集合をソートしたもの作成
sorted_keys.sort(key=lambda n0: len(n0)) # さらにkeyの長さでソート
for key in sorted_keys:  # 上記の順に key, orders[key] を print
    print(key)
    print(orders[key])
```



安定ソート (stable sort)
評価値が同じ要素の並び順を変えないソートのこと

ライブラリと計算量（C++）

ライブラリは、
「用途に応じて効率的に」
実装されています。

- vector: 配列を用いた実装
  - 各要素へのアクセスは、要素数によらず一定
  - 要素の検索や先頭要素の削除は要素数に応じた時間必要
  - 配列のコピーも要素数に応じたコスト：
    - 長い vector は値渡しではなく参照渡しにしよう

Java: ArrayList, python: list も同様

Java, python のオブジェクトは
基本参照を用いて扱う

- sort: 要素数 n に対して log(n) x n に比例する計算量
  - $\log(2^{10}) = 10$, $\log(2^{20}) = 20$ なので、要素数が大きく増えても log 部分の増加は小さい （参考：ソートのアルゴリズム）

Java, python も同様

- map: 要素の検索・追加・削除は log(n) に比例する計算量（n: 要素数）（赤黒木のアルゴリズム）

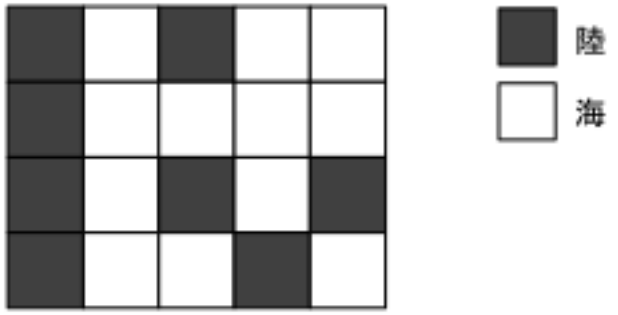
一般にソート系は log(n),
Hash 系は 一定コスト で
挿入可能

ライブラリを利用する利点

- vector なんて、大きい配列と要素数があれば作れるじゃん！
  - そのとおり！見栄えや手間の問題です。
    でも、「見やすい」「楽」「バグがない」って重要なこと。
  - ヤヤコシイことを
    - 関数に分離
    - ライブラリにお任せ

問題例：

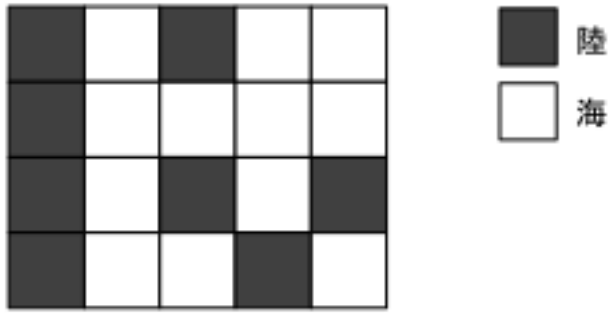
- 問題：島の数を数える
  - 斜めもつながっている
  - 出典：ACM ICPC 2009 国内予選 ProblemB



- 解法例
  - 島を一つ探す
  - 「たどれる」ところを「たどる」
  - 次の島を探して、同じ処理を。

問題例（続，C++）

- 「たどれるところをたどる」って？
  - 最初の陸地を「袋」に入れる
  - 「袋」から一つ取り出して、周囲の陸地をstackに詰める。
  - 「袋」に詰めた陸地は、何度も調べないよう沈めましょう
- point は complex を使ってます



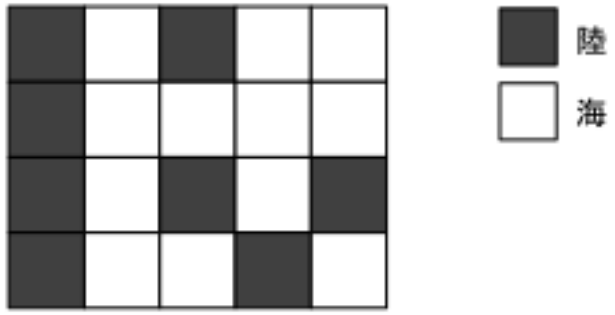
program@github

```
stack<point> searching;
while(numLand > 0) {
  numIland++;
  auto one = searchOne();
  searching.push(one);
  delLand(one);
  while (!searching.empty()) {
    auto one = searching.top();
    searching.pop();

    for(auto & direct1: directions) {
      auto next = one + direct1;
      if (rangeOut(next)) continue;
      if (c[next.imag()][next.real()] > 0) {
        delLand(next);
        searching.push(next);
      }
    }
  }
}
```

問題例（続, Java）

- 「たどれるところをたどる」って？
  - 最初の陸地を「袋」に入れる
  - 「袋」から一つ取り出して、周囲の陸地をstackに詰める。
  - 「袋」に詰めた陸地は、何度も調べないよう沈めましょう
- point は awt の Point を利用



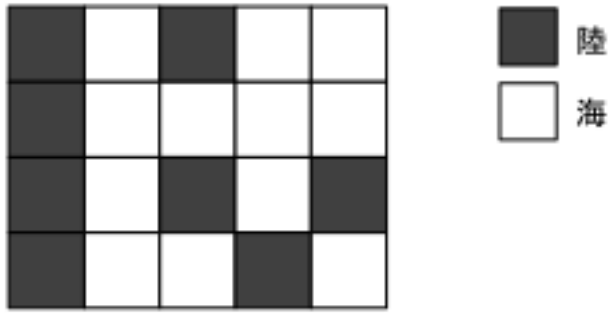
program@github

```
int numIlands = 0;
Stack<Point> searching = new
while(numLands>0) {
    numIlands++;
    Point first = searchOne();
    searching.push(first);
    delLand(first);
    while(!searching.empty()) {
        Point one = searching.pop();
        for(Point d: directions) {
            Point next =
              new Point(one.x+d.x, one.y+d.y);
            if(rangeOut(next)) continue;
            if(c[next.y][next.x] > 0) {
                delLand(next);
                searching.push(next);
            }
        }
    }
}
```

問題例（続, python）

program@github

- 「たどれるところをたどる」って？
  - 最初の陸地を「袋」に入れる
  - 「袋」から一つ取り出して、周囲の陸地をstackに詰める。
  - 「袋」に詰めた陸地は、何度も調べないよう沈めましょう
- point は list を利用




```
while self.num_lands > 0:
    num_ilands += 1
    one = self.search_one()
    self.del_land(one[0], one[1])
    searching = deque([one])
    while searching:
        x, y = searching.pop()
        for direct1 in directions:
            dx, dy = direct1
            nx, ny = [x + dx, y + dy]
            if self.range_check(nx, ny):
                continue
            if self.c[ny][nx] > 0:
                self.del_land(nx, ny)
                searching.append([nx, ny])
return num_ilands
```

Stack, Queue, 優先度 Queue (C++)

全部リストの一種

- Stack
  最後に入れたものから出す
- Queue
  最初に入れたものから出す
- Priority Queue
  デカいものから出す

```
stack<int> stack;   // 作成
stack.push(val);    // 要素追加
stack.top();        // 要素アクセス
stack.pop();        // 要素削除
```

```
queue<int> queue;   // 作成
queue.push(val);    // 要素追加
queue.front();      // 先頭アクセス
queue.pop();        // 要素削除
```

```
priority_queue<int> pq; // 作成
pq.push(val);   // 要素追加
pq.top();       // 要素アクセス
pq.pop();       // 要素削除
```

Int 版program@github
complex 版program@github

Stack, Queue, 優先度 Queue (Java)

全部リストの一種

- Stack
  最後に入れたものから出す
- Queue
  最初に入れたものから出す
- Deque: どちらにも使える
- Priority Queue
  小さいもの順で取り出す

```
Stack<Integer> stack = new Stack<>();
stack.push(val);  // 要素追加, add(v) も可
stack.peek()      // 要素アクセス
stack.pop();      // 要素取得＆削除
```

```
Queue<Integer> queue = new Deque<>();
queue.add(val);// 要素追加
queue.peek();  // 先頭アクセス, element()も可
queue.poll();  // 要素削除, remove() も可
```

```
PriorityQueue<Integer> pq = new ...;
pq.add(val); // 要素追加
pq.peek();   // 要素アクセス, element()も可
pq.poll();   // 要素削除, remove() も可
```

Int 版program@github
Point 版program@github

Deque, Heapq (python)

全部リストの一種

- Deque
  - 先頭or最後に要素を追加
  - 先頭or最後から要素を削除
- HeapQueue
  小さいもの順で取り出す
  - 要素にリストを使うと、
    複数要素を辞書順にソート

```
deque = deque()
deque.append(val)     # 最後に要素追加
deque.appendLeft(val)# 先頭に要素追加
deque.pop()           # 最後の要素取得＆削除
deque.popLeft()       # 先頭の要素取得＆削除
```

```
priq = []    # 作成するのは普通のリスト
heappush(priq, val) # 要素追加
val = heappop(priq) # 要素取得＆削除
# 優先度に2指標(abs(val), val.real)を利用したい場合
priq2 = []
heappush(priq2, [abs(val), val.real, val])
pri0, pri1, val = heappop(priq2)
```

Int 版program@github
complex 版program@github