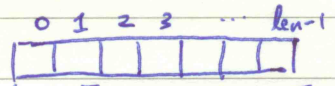
* data structure

construct <----> destruct

~~get~~ ⇒ find

put

remove ) dynamic maintenance




有编号的抽屉.

* C/C++ (build-in) array

Fixed Len Array

⓪ construct destruct

C: int arr[] = (int*) malloc(sizeof(int) * len); free(arr);

C++: int arr[] = new arr[len]; delete[] arr;

initialized to 0 ↓ uninitialized

may need other initialization steps to EMPTY

① int findAtIndex(int idx) {
    return arr[idx];
}

② void putAtIndex(int idx, int& value) { // throw away original if non-empty
    arr[idx] = value;
}

→ fast "random" access efficiently (indexed)

by translating arr[idx] to

*(arr + idx)

start idx steps, each of length sizeof(int)

③ put(value) or insert(value)

```
for i ← 0 to len-1
    if findAtIndex(i) = EMPTY
        putAtIndex(i, value)
        return
// error (full) if here.
```

④ removeAtIndex(idx)

```
arr[idx] = EMPTY
```

* ConsecutiveArray

a FixedLenArray such that all non-EMPTY elements are in arr[0], arr[1], ..., arr[end-1]

# elements

⓪ construct:

from nothing: FixedLenArray.construct()

from FixedLenArray:

"compress" EMPTY spaces

① findAtIndex: trivial

② putAtIndex(idx, value)

```
if (idx < end)
    ...
else
    // something needs to be done
```

③ insert(value):

insert at "end" and end ← end+1, perhaps

④ removeAtIndex(idx):

roughly, swap with arr[end] and then remove, perhaps

Consecutive Array "easier" insert by keeping "end"
slightly harder putAtIndex and removeAtIndex

* array for storing top game entries

```
class Record {
public:
    String name;
    int score;
};
```

need:

① often: list top 20 players in order

① often: update list by "inserting" new record & removing old one (lower)

② seldom: remove some records

* if using FixedLenArray of size 20

① findFirst, findSecond, ..., find19th, find20th ?

same for using ConsecutiveArray

OrderedArray:

a ConsecutiveArray such that

arr[0] ≥ arr[1] ≥ arr[2] ≥ ... ≥ arr[end-1]

• score

① construct

from nothing: ConsecutiveArray.construct()

from ConsecutiveArray: selectionSort()

(new) listOrderly()

```
for i ← 0 to len-1
    print arr[i]
```

① findAtIndex : trivial

② putAtIndex : VERY restricted

③ insert(value) :

先"找到"

從頭 for i ← 0 to end-1
    if arr[i] < value
        插隙

從尾 for i ← end-1 to 0
    if arr[i] > value
        插隙

找到 + 插隙 can be combined here (see textbook)

④ removeAtIndex(idx) :

"補位"

```
for i ← idx+1 to end-1
    arr[i-1] ← arr[i]
end ← end-1
```

* about "找到" findIndexOf(value)
findRecordOf(key)
score here

naive: Sequential Search len-1 non-Consecutive

```
for i ← 0 to end-1
    if arr[i].score = key
        return arr[i]
return FAIL
```

Consecutive

Ordered 做"之步

```
else if arr[i].score < key
    return FAIL
```

No. Date. 2-5

* 偷吃步 ① for Ordered

if arr[i].score < key 後面都不用看了

偷吃步 ② for Ordered

if arr[i] > key 前面都不用看了

[left, right) ← [0, end)

while left < right // not 空空如也

pick i

if arr[i] = key

return arr[i] // 找到

else if arr[i] < key

find in ~~前面~~ // ①

[0, i) : still Consecutive & Ordered

right ← i

left

else if arr[i] > key

find in ~~後面~~ // ②

[i+1, ~~end~~) : still Consecutive & Ordered

left ← i+1

right

return FAIL

* "pick i" : ambiguous (violate definiteness)

- pick i within [left, right) better, but how?
- pick i uniformly randomly within [left, right) ?
- pick i ← left ?

if lucky, 找到 or 偷一大步

else 沒偷到

* Binary Search

pick $i \leftarrow \left\lfloor \frac{left + right}{2} \right\rfloor$ (usually named mid)

claim: after executing "pick i" $\lceil \log_2 end \rceil + 1$ times
right − left < 1 (끝날 수도 있)

lemma: every non-found updating after "pick i" decreases right − left by at least half

case ①: right − left becomes

$$\left\lfloor \frac{left+right}{2} \right\rfloor - left \leq \frac{left+right}{2} - left$$

$$= \frac{right - left}{2}$$

case ②: right − left becomes

$$right - \left\lfloor \frac{left+right}{2} \right\rfloor - 1$$

$$\leq right - \left( \frac{left+right}{2} - 1 \right) - 1$$

$$= \frac{right - left}{2}$$

so after $\lceil \log_2 end \rceil + 1$ times of "pick i"

$$right - left \leq \frac{end - 0}{2^{\lceil \log_2 end \rceil + 1}}$$

$$\leq \frac{end}{2^{\log_2 end + 1}}$$

$$\leq \frac{1}{2}$$

$$< 1$$

⇒ Binary search terminates after at most $\lceil \log_2 end \rceil + 1$ iterations □