Dynamic Array Queue and Deque
Queues

```c
int isEmpty();
void addBack(TYPE val); // Add value at end of queue.
TYPE front(); // Get value at front of queue.
void removeFront(); // Remove value at front.
```

remove from the front

add to the back
What nodes are reachable from A?
Queue with Dynamic Array

```c
int isEmpty();
void addBack(TYPE val); // Add value at end of queue.
TYPE front(); // Get value at front of queue.
void removeFront(); // Remove value at front.
```

Removal from front is expensive!
Deque (Double Ended Queue) ADT

void addFront(TYPE val);
void removeFront();
TYPE front();
void addBack(TYPE val);
void removeBack();
TYPE back();
Deque Application

- Finite Length Undo
- Palindrome Checker (e.g. radar)
Let the partially filled block “float” & “wrap”

Allow the starting index, beg, to “float” around the underlying array! It’s no longer confined to index 0

Logical index

beg=3

size

0 1 2 3 4 5 6 7

Absolute index

How??

Keep track of the data, arranged circularly and convert all indices into proper indices for the actual array!

Circular Buffer
struct ArrDeque {
    TYPE *data; /* Pointer to data array. */
    int size; /* Number of elements in collection. */
    int beg; /* Index of first element. */
    int cap; /* Capacity of array. */
};

void initArrDeque(struct ArrDeque *d, int cap) {
    d->data = malloc(cap * sizeof(TYPE));
    assert(d->data != 0);
    d->size = d->beg = 0;
    d->cap = cap;
}
Adding/Removing from Back

**Add**

- beg
- size

**Remove**

- beg
- size

---

Adding/Removing from Back
Adding/Removing from Front

**Add**

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Remove**

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

beg | size

beg | size

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

beg | size

beg | size

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

beg | size

beg | size
Elements can wrap around from beg to end
• Calculate offset: add logical (element) index to start \( (\text{beg}) \)

\[
\text{offset} = \text{beg} + \text{logicalIndex}; \quad /* \text{logIndex} = 3, \text{offset} = 9 */
\]

• If larger than or eq to capacity, subtract capacity

\[
\text{if (offset} \geq \text{cap)}
\]

\[
\text{absoluteIndex} = \text{offset} - \text{cap};
\]

• Alternatively, use mod:

\[
/* \text{Convert logical index to absolute index.} */
\]

\[
\text{absIdx} = (\text{logicalIdx} + \text{beg}) \% \text{cap};
\]
Can we simply copy the elements to a larger array?

Have to be careful because the wrapping is dependent on the ‘capacity’
Key Changes from Dynamic Array

- Keep track of floating data with \textit{beg} & \textit{size}
- Wrap \textit{beg} as necessary
- Whenever accessing a logical index, convert to absolute first
- On resize, copy to start of the new array and reset \textit{beg}
void addFrontDynArr(DynArr *v, TYPE val){
    if (v->size >= v->capacity)
        _dynArrSetCapacity(v, 2*v->capacity);
    v->beg = v->beg - 1;
    if(v->beg < 0)
        v->beg = v->capacity-1;
    v->data[_absoluteIndex(v, 0)] = val;
    v->size++;
}

Always make sure you have space

Wrap beg as necessary

Don’t forget to increment size

front is at logical index = 0

convert to proper absolute index

Update beg

Update beg

Update beg
Let’s look at some code...

```c
void addFrontDynArr(DynArr *v, TYPE val) {
  if (v->size >= v->capacity)  
    dynArrSetCapacity(v, 2*v->capacity);
  v->beg = v->beg - 1;
  if (v->beg < 0)  
    v->beg = v->capacity - 1;
  v->data[absoluteIndex(v, 0)] = val;
  v->size++;
}
```

Always make sure you have space

Wrap beg as necessary

Don’t forget to increment size

Full implementation is in dynamicArrayDeque.c

convert to proper absolute index

front is at logical index = 0

}
## Operations Analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>Best</th>
<th>Worst</th>
<th>Ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddBack</td>
<td>1</td>
<td>n</td>
<td>1+</td>
</tr>
<tr>
<td>RemoveBack</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AddFront</td>
<td>1</td>
<td>n</td>
<td>1+</td>
</tr>
<tr>
<td>RemoveFront</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>