รายการ

(ArrayList & LinkedList)

หัวข้อ

- นิยามรายการ และอินเตอร์เฟส List
- การสร้างรายการด้วยอาเรย์
- การสร้างรายการด้วยการโยง
  - โยงเดี่ยวแบบไม่วนที่มีปมหัว
  - โยงคู่แบบวนที่มีปมหัว
- การสร้างเวกเตอร์มากเลขศูนย์ด้วยรายการ

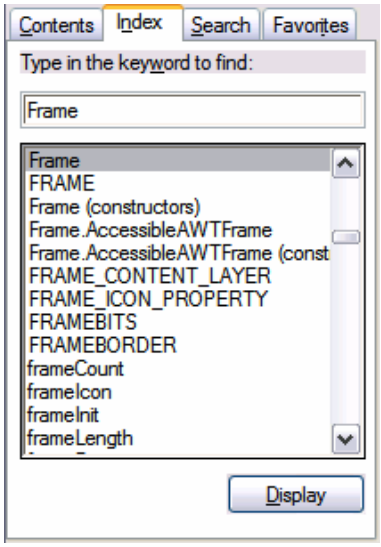
รายการ (List)

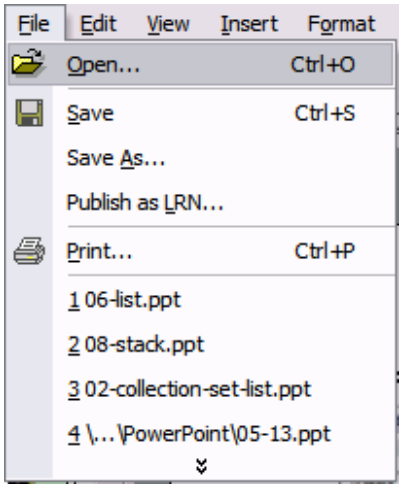
- List ก็เป็น Collection อย่างหนึ่ง
- เก็บแบบมีอันดับ ข้อมูลแต่ละตัวมีหมายเลขกำกับ

$$< a_0, a_1, a_2, ..., a_{n-1} >$$

```
public interface List extends Collection {
  public void add(int index, Object e);
  public void remove(int index);
  public Object get(int index);
  public void set(int index, Object e);
  public int indexOf(Object e);
}
```


ตัวอย่างรายการ






```
<"SU", "MO", "TU", "WE", "TH", "FR", "SA">
```

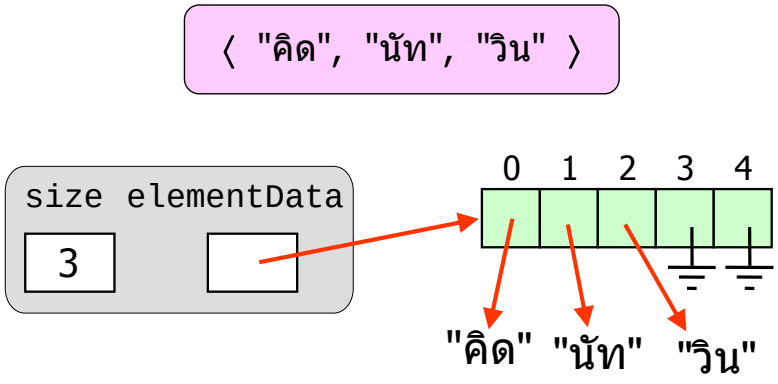
$f(x) = 2x^5 + 4x^3 - 6x + 9$ $< (2,5), (4,3), (-6,1), (9,0) >$

ตัวอย่างการใช้งาน

```
List x = new ArrayList(10);     ⟨ ⟩
x.add(0,"A");                   ⟨ "A" ⟩
x.add("B");                     ⟨ "A", "B" ⟩
x.add(0,"C");                   ⟨ "C", "A", "B" ⟩
x.set(2,"D");                   ⟨ "C", "A", "D" ⟩
int i = x.indexOf("A");
x.add(i,"Z");                   ⟨ "C", "Z", "A", "D" ⟩
x.remove(2);                    ⟨ "C", "Z", "D" ⟩
for(int i=0; i<x.size(); i++)
  System.out.println(x.get(i));
```


การสร้างรายการด้วยอาเรย์

- สร้างอาเรย์เก็บข้อมูลเริ่มตั้งแต่ช่องที่ 0
- มีตัวแปร size เก็บจำนวนข้อมูล
- $< a_0, a_1, a_2, ..., a_{n-1} >$ เก็บข้อมูลไว้ที่ elementData[0], ..., elementData[n – 1] ตามลำดับ



ArrayList : เมท็อดของ Collection

```
public class ArrayList implements List {
  private Object[] elementData;
  private int size;
  public ArrayList(int cap) {
    elementData = new Object[cap];
    size = 0;
  }
  public int size() { return size; }
  public boolean isEmpty() { return size == 0; }
  public boolean contains(Object e) {
    return indexOf(e) != -1;
  }
  public void add(Object e) {
    add(size, e);
  }
  public void remove(Object e) {
    int i = indexOf(e);
    if (i >= 0) remove(i);
  }
```


indexOf, get, set

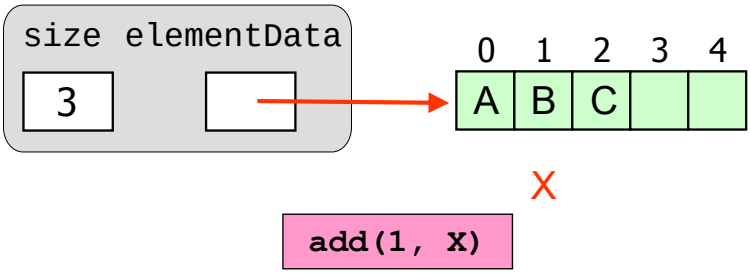
```
public class ArrayList implements List {
  ...
  public int indexOf(Object e) {
    for(int i=0; i<size; i++)
      if (elementData[i].equals(e)) return i;
    return -1;
  }
  public Object get(int index) {
    return elementData[index];
  }
  public void set(int index, Object e) {
    elementData[index] = e;
  }
```

$O(n)$

$\Theta(1)$

$\Theta(1)$

add(index, e)

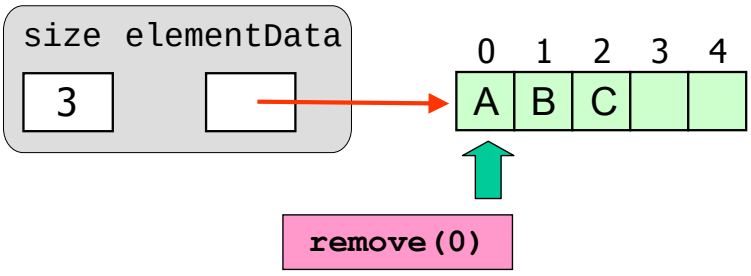



```
public void add(int index, Object e) {
  ensureCapacity(size+1);
  for(int i=size; i>index; i--) {
    elementData[i] = elementData[i-1];
  }
  elementData[index] = e;
  size++;
}
```

$O(n)$

add(0,e) ช้า add(size,e) เร็ว

remove(index)

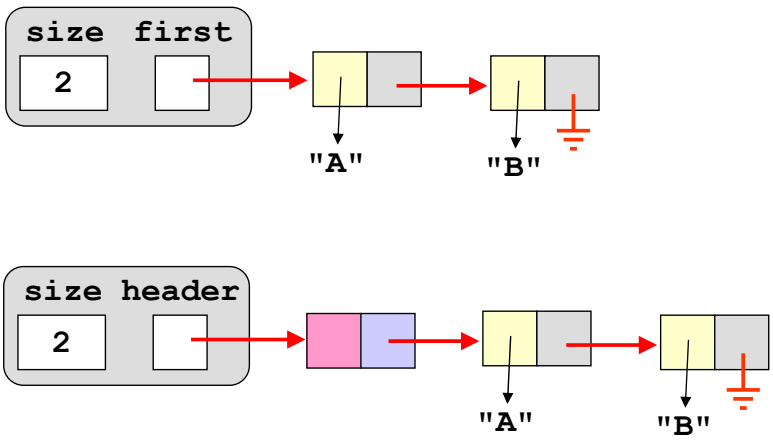



```
public void remove(int index) {
  for(int i=index+1; i<size; i++) {
    elementData[i-1] = elementData[i];
  }
  size--;
  elementData[size] = null;
}
```

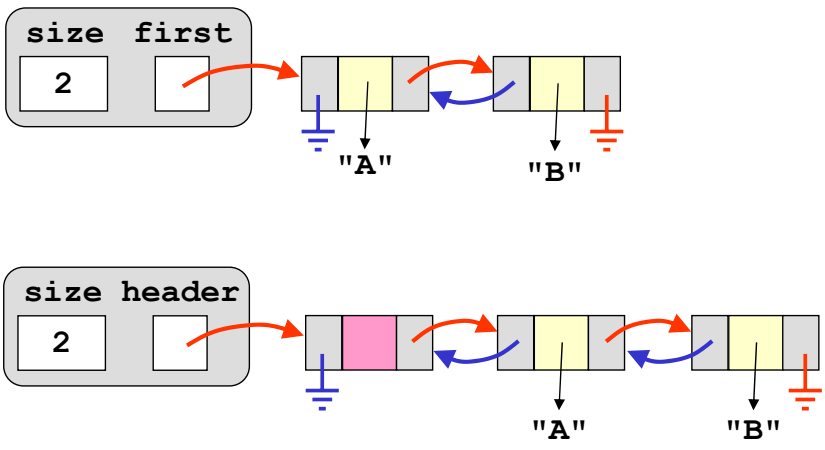
$O(n)$

remove(0) ช้า remove(size-1) เร็ว

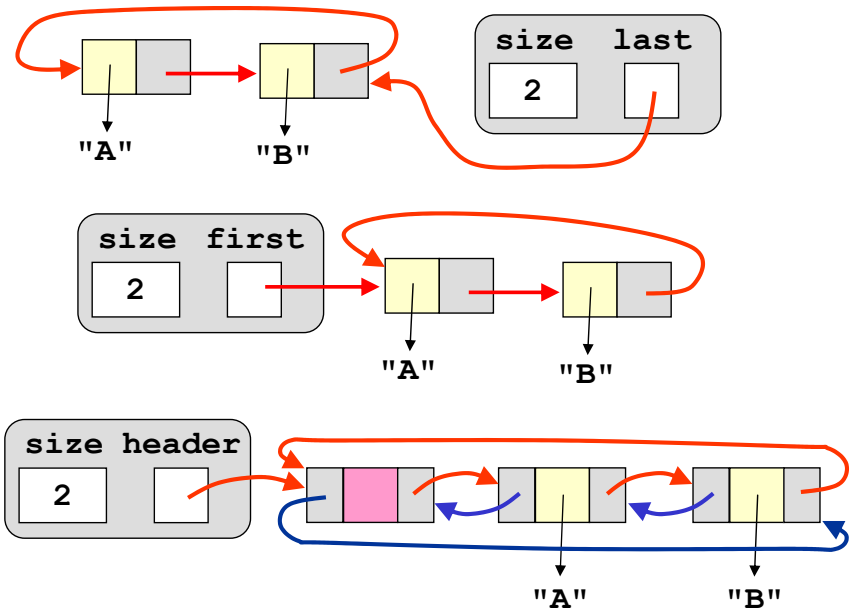
การสร้างแบบโยงเดี่ยว (singly linked)



การสร้างแบบโยงคู่ (doubly linked)



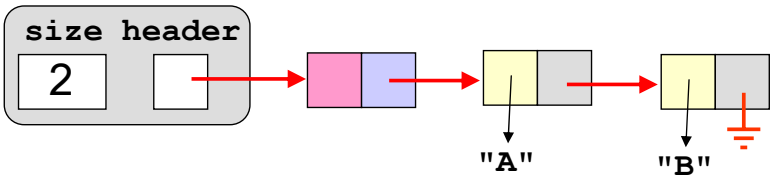
การสร้างแบบโยงวน (circular)



รายการโยงเดี่ยวแบบไม่วนที่มีปมหัว

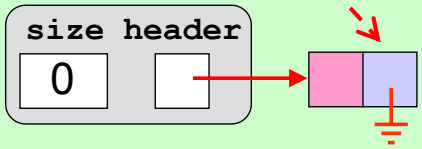
```
public class SinglyLinkedList implements List {
  private static class LinkedNode {
    Object element;
    LinkedNode next;
    LinkedNode(Object e, LinkedNode n) {
      this.element = e;
      this.next = n;
    }
  }
  private LinkedNode header;
  private int size;
  ...
```

Singly linked list with header



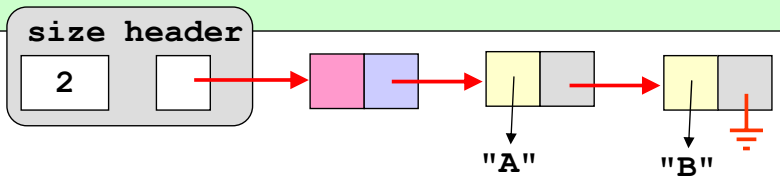
SinglyLinkedList

```
public class SinglyLinkedList implements List {
  ...
  private LinkedNode header = new LinkedNode(null,null);
  private int size = 0;

  public SinglyLinkedList() { }
  public boolean isEmpty() {
    return header.next == null;  // size == 0
  }
  public int size() {
    return size;
  }
  ...
```



indexOf(e), contains(e)

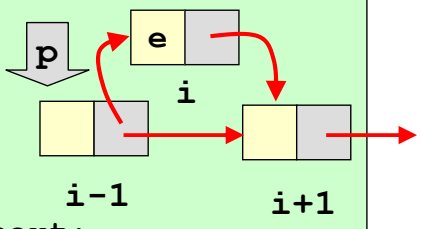
```
public class SinglyLinkedList implements List {
  ...
  public int indexOf(Object e) {
    LinkedNode q = header.next;
    for (int i=0; i<size; i++) {
      if (q.element.equals(e)) return i;
      q = q.next;
    }
    return -1;
  }
  public boolean contains(Object e) {
    return indexOf(e) >= 0;
  }
  ...
```



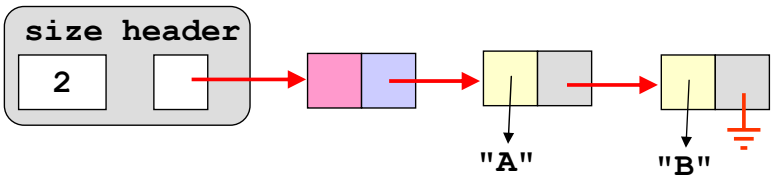
add(e), add(i, e)

```
public void add(Object e) {
  add(size, e);
}
public void add(int i, Object e) {
  LinkedNode p = nodeAt(i-1);
  p.next = new LinkedNode(e, p.next);
  ++size;
}
private LinkedNode nodeAt(int i) {
  LinkedNode p = header;
  for (int j = -1; j < i; j++) p = p.next;
  return p;
}
```

ขอปมที่อยู่ตำแหน่ง i – 1

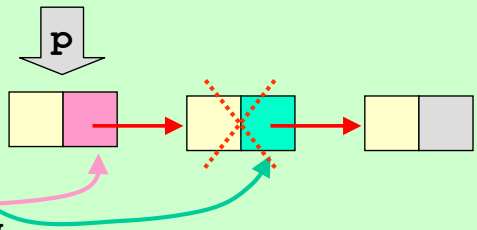


i เป็น –1 จะคืน header



remove(e), remove(i)

```
private void removeAfter(LinkedNode p) {
  if (p.next != null) {
    p.next = p.next.next;
    --size;
  }
}
public void remove(Object e) {
  LinkedNode p = header;
  while (p.next != null && !p.next.element.equals(e))
    p = p.next;
  removeAfter(p);
}
public void remove(int i) {
  LinkedNode p = nodeAt(i-1);
  removeAfter(p);
}
```



วิ่งหาปมก่อน
หน้าปมที่เก็บ e

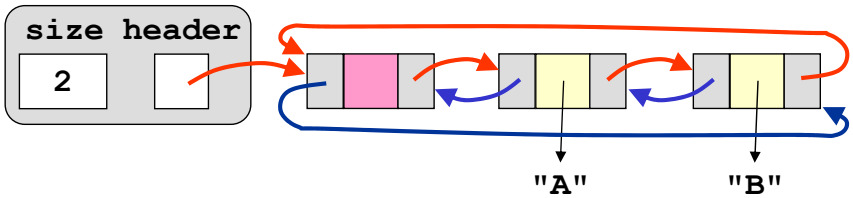
get(i), set(i, e)

```
public Object get(int i) {
  return nodeAt(i).element;
}
public void set(int i, Object e) {
  nodeAt(i).element = e;
}
```


รายการโยงคู่แบบวนที่มีปมหัว

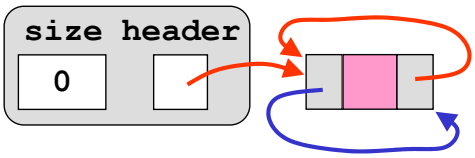
```
public class LinkedList implements List {
  private static class LinkedNode {
    Object element;
    LinkedNode prev, next;
    LinkedNode(Object e, LinkedNode p, LinkedNode n) {
      this.element = e;
      this.prev = p;
      this.next = n;
    }
  }
  private LinkedNode header;
  private int size;
  ...
```

Circular doubly linked list with header



constructor

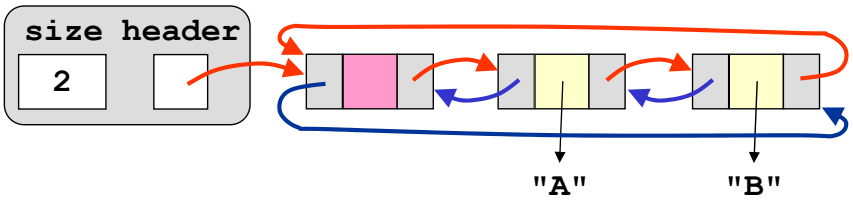
```
public class LinkedList implements List {
  ...
  private LinkedNode header;
  private int size;

  public LinkedList() {
    header = new LinkedNode(null, null, null);
    header.prev = header.next = header;
  }
```



เมท็อดที่เหมือนกับ SinglyLinkedList

- size()
- isEmpty()
- indexOf(e)
- contains(e)
- nodeAt(i)
- get(i)
- set(i,e)

เมท็อดที่ไม่เหมือน
(เพราะต้องจัดการ prev ด้วย)
- add(e)
- add(i,e)
- remove(e)
- remove(i)



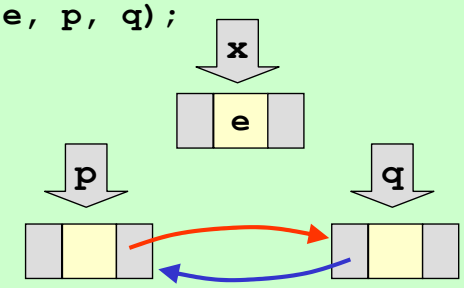
add(e), add(i, e)

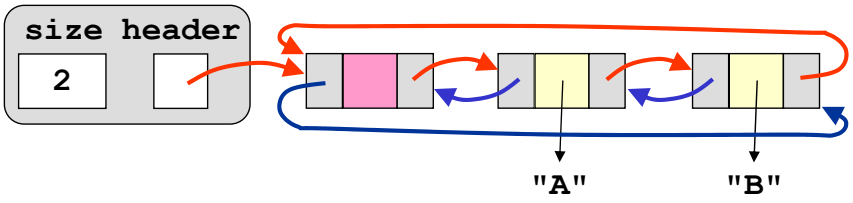
```
private void addBefore(LinkedNode q, Object e) {
  LinkedNode p = q.prev;
  LinkedNode x = new LinkedNode(e, p, q);
  p.next = q.prev = x;
  ++size;
}
public void add(Object e) {
  addBefore(header, e);
}
public void add(int i, Object e) {
  addBefore(nodeAt(i), e);
}
```

Θ(1)

Θ(1)

O(*n*)





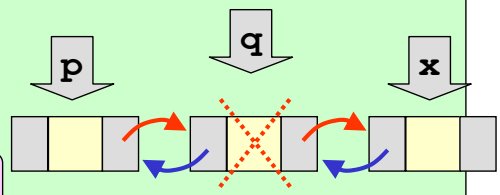
remove(e), remove(i)

```
private void removeNode(LinkedNode q) {
  LinkedNode p = q.prev;
  LinkedNode x = q.next;
  p.next = x;
  x.prev = p;
  --size;
}
public void remove(int i) {
  removeNode(nodeAt(i));
}
public void remove(Object e) {
  LinkedNode q = header.next;
  while (q != header) {
    if (q.element.equals(e)) { removeNode(q); break; }
    q = q.next;
  }
}
```

Θ(1)

O(*n*)

O(*n*)



เวกเตอร์มากเลขศูนย์ (Sparse Vector)

- นิยามของเวกเตอร์มากเลขศูนย์
- การสร้างเวกเตอร์มากเลขศูนย์ด้วยรายการ
- รายละเอียดของบริการต่าง ๆ

เวกเตอร์มากเลขศูนย์

- ต้องการใช้เวกเตอร์ เช่น (0,3,0,0,0,0,4)
- ใช้อาเรย์ double[ ] x = {0,3,0,0,0,0,4};
- ถ้าเวกเตอร์มีจำนวนส่วนใหญ่เป็น 0
- เก็บเป็นรายการของตัวที่ไม่ใช่ 0 ⟨(1,3),(6,4)⟩

```
SparseVector v1 = new SparseVector(7);
v1.set(1,3);                    // (0,3,0,0,0,0,0)
v1.set(6,4);                    // (0,3,0,0,0,0,4)
SparseVector v2 = v1.add(v1); // (0,6,0,0,0,0,8)
v2.set(2,3);                    // (0,6,3,0,0,0,8)
double d = v1.dot(v2);
```

```
   v1 = (0,3,0,0,0,0,4)
   v2 = (0,6,3,0,0,0,8)
v1·v2 =   18   +  32 = 50
```

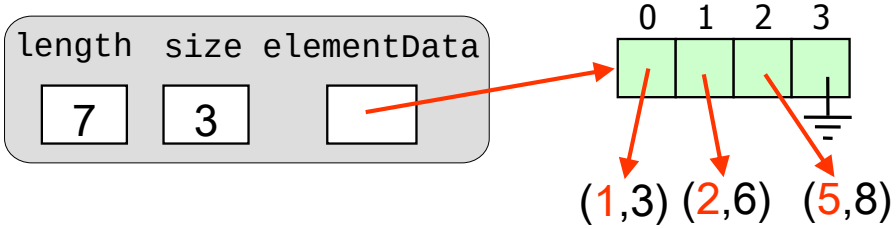
บริการของ SparseVector

```
public class SparseVector {
  public int length() {...}
  public double get(int i) {...}
  public void set(int i, double x) {...}
  public SparseVector add(SparseVector v) {...}
  public double dot(SparseVector v) {...}
  public SparseVector multiply(double x) {...}
}
```


สร้างด้วยแถวลำดับ

- elementData เก็บแถวลำดับ
  - แต่ละช่องเก็บคู่ลำดับ (index, value) เก็บเรียง index จากน้อยไปมาก
- size เก็บจำนวนคู่ลำดับในรายการ
- length เก็บความยาว vector

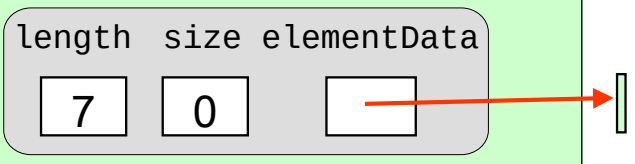
```
SparseVector v1 = new SparseVector(7);
v1.set(1,3);                    // (0,3,0,0,0,0,0)
v1.set(5,8);                    // (0,3,0,0,0,8,0)
v1.set(2,6);                    // (0,3,6,0,0,8,0)
```



SparseVector

```
public class SparseVector {
  private static class Element {
    int index;
    double value;
    Element(int i, double v) {
      this.index = i;  this.value = v;
    }
  }
  private int size;
  private int length;
  private Element[] elementData;
  public SparseVector(int length) {
    this.elementData = new Element[0];
    this.size = 0;
    this.length = length;
  }
  public int length() { return length; }
```

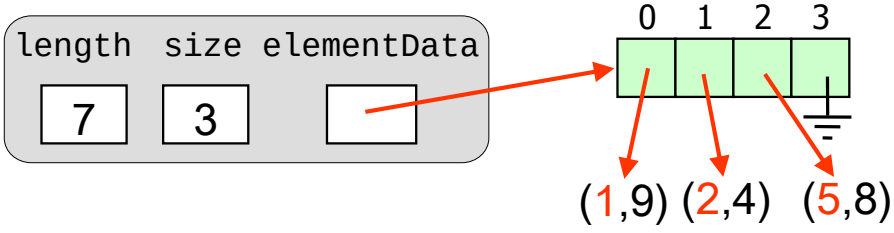
เก็บคู่ลำดับ (index, value)



SparseVector : get

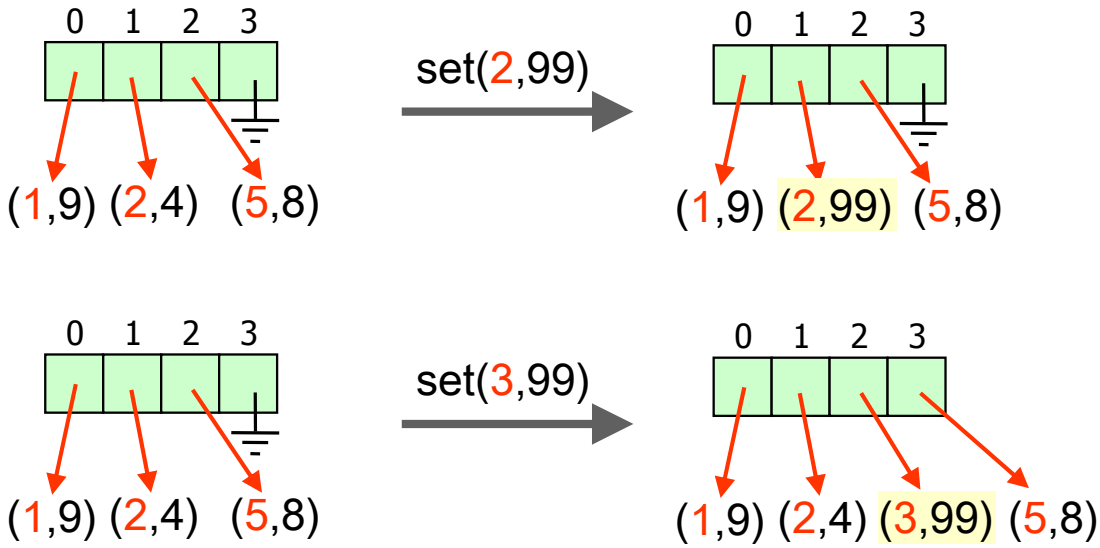
```
public class SparseVector {
  ...
  public double get(int index) {
    for(int i=0; i<size; i++) {
      if (elementData[i].index == index)
        return elementData[i].value;
      if (elementData[i].index > index) break;
    }
    return 0.0;
  }
```

O(n)



SparseVector : set

```
public class SparseVector {
  ...
  public void set(int index, double value) {
```



SparseVector : set

```
  public void set(int index, double value) {
    int i = 0;
    for ( ;i<size; i++ )
      if (elementData[i].index >= index) break;
    if (i<size && elementData[i].index == index)
      elementData[i].value = value;
    else
      add(i, index, value);
  }
  void add(int i, int index, double value) {
    if (value != 0) {
      ensureCapacity(size+1);
      for (int k=size; k>i; k--)
        elementData[k] = elementData[k-1];
      elementData[i] = new Element(index, value);
      ++size;
    }
  }
```

$O(n)$

SparseVector : dot

```
public double dot(SparseVector v2) {
  SparseVector v1 = this;
  double r = 0;
  for (int i = 0; i<v1.length(); i++)
    r += v1.get(i) * v2.get(i);
  return r;
}
```

```
v1 = (0,3,0,0,1,4,0)
v2 = (0,6,3,0,0,3,0)
r = 0*0 + 3*6 + 0*3 +
    0*0 + 1*0 + 4*3 + 0*0
```

- เสีย O(n) ในแต่ละครั้งที่ get
- ต้องเรียก m ครั้ง (m คือความยาวเวกเตอร์)
- รวมใช้เวลา O(nm)

SparseVector : dot

```
public double dot(SparseVector v2) {
  SparseVector v1 = this;
  double r = 0;
  int i1 = 0, i2 = 0;
  while (i1 < v1.size && i2 < v2.size) {
    Element e1 = v1.elementData[i1];
    Element e2 = v2.elementData[i2];
    if      (e1.index < e2.index) i1++;
    else if (e1.index > e2.index) i2++;
    else {
      r += e1.value * e2.value;
      i1++; i2++;
    }
  }
  return r;
}
```

O(*n*)

```
v1 = (0,3,0,0,0,4,0)
v2 = (0,6,3,0,0,8,0)
v1·v2 =   18   +  32 = 50
```

SparseVector : add

```
public SparseVector add(SparseVector v2) {
  SparseVector v1 = this;
  SparseVector v3 = new SparseVector(v1.length());
  for (int i = 0; i<v1.length(); i++)
    v3.set(i, v1.get(i) + v2.get(i));
  return v3;
}
```

```
v1 = (0,3,0,0,1,4,0)
v2 = (0,6,3,0,0,3,0)
v3 = (0,9,3,0,1,7,0)
```

- เสีย O(n) ในแต่ละครั้งที่ get
- set ที่ปลายเวกเตอร์ เสีย O(n)
- ต้องเรียก m ครั้ง (m คือความยาวเวกเตอร์)
- รวมใช้เวลา O(nm)

$O(n)$

```
public SparseVector add(SparseVector v2) {
  SparseVector v1 = this;
  SparseVector v3 = new SparseVector(v1.length());
  int i1 = 0, i2 = 0, i3 = 0;
  while (i1 < v1.size && i2 < v2.size) {
    Element e1 = v1.elementData[i1], e2 = v2.elementData[i2];
    if (e1.index < e2.index)
      {v3.add(i3++, e1.index, e1.value); i1++;}
    else if (e1.index > e2.index)
      {v3.add(i3++, e2.index, e2.value); i2++;}
    else
      {v3.add(i3++, e1.index, e1.value+e2.value); i1++;i2++;}
  }
  while (i1 < v1.size) {
    Element e1 = elementData[i1++];
    v3.add(i3++, e1.index, e1.value);
  }
  while (i2 < v2.size) {
    Element e2 = elementData[i2++];
    v3.add(i3++, e2.index, e2.value);
  }
  return v3;
}
```

สรุป

- รายการคือที่เก็บข้อมูลที่ข้อมูลแต่ละตัวมีอันดับ
- สร้าง ArrayList คล้ายกับ ArrayCollection
- สร้าง SinglyLinkedList คล้ายกับ LinkedCollection
  - โยงเดี่ยว ไม่วน มีปมหัว
- สร้าง LinkedList
  - โยงคู่ วน มีปมหัว
- การใช้รายการเพื่อสร้างเวกเตอร์มากเลขศูนย์