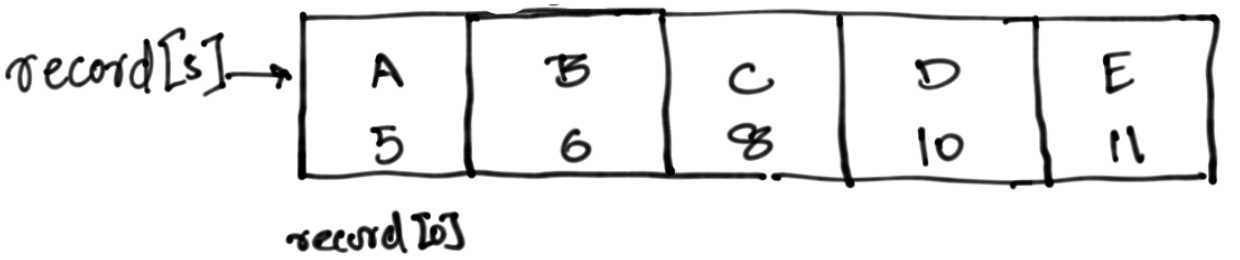
Data - Structure Question

(1) Assume that you are the owner of a company that has 5 employees.

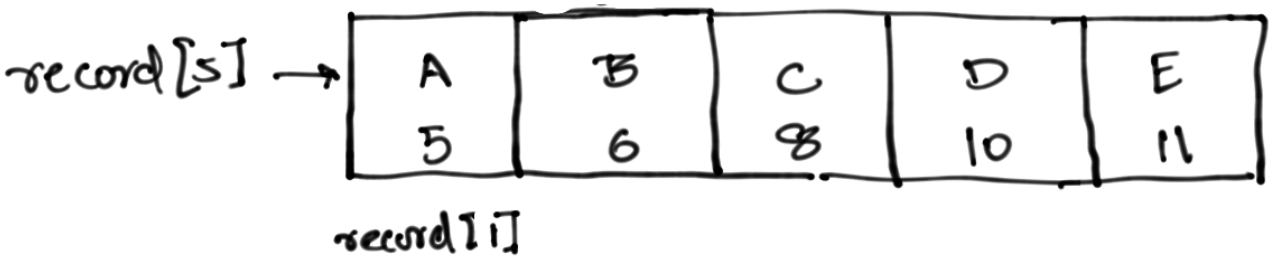
(2) Each employee is represented by {name, salary}

Which data-structure (that you know) will you use to represent these 5 employees?

Arrays

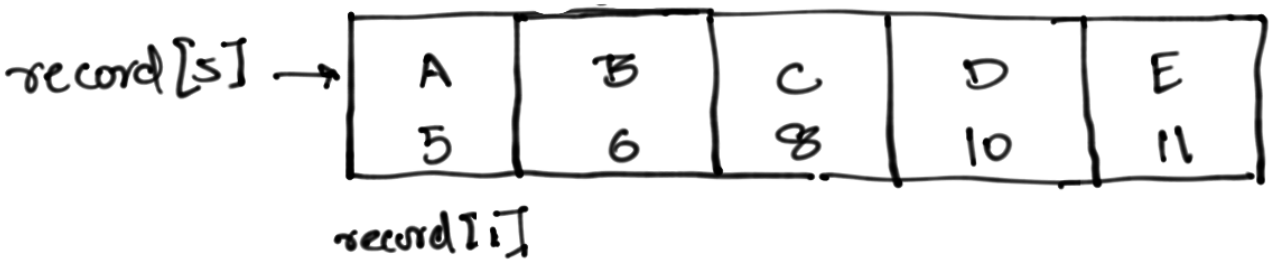


Arrays



Suppose a new employee {F, 10} joins the company. How will you update your data-structure?

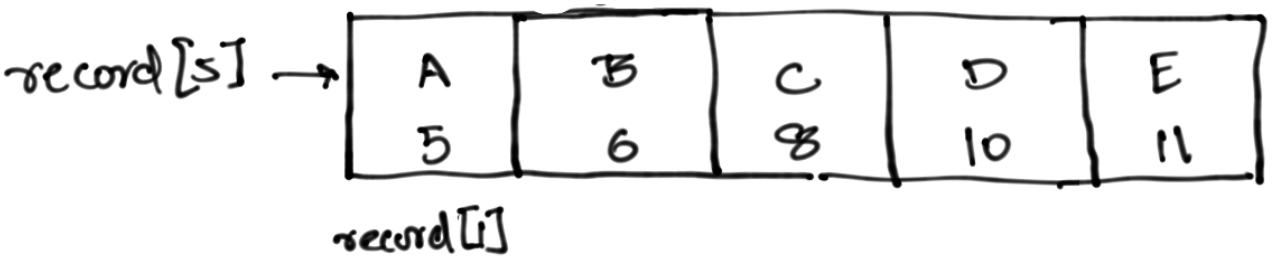
Arrays



Suppose a new employee {F, 10} joins the company. How will you update your data-structure?

Add {F, 10} @ record[6]

Arrays



Suppose a new employee {F, 10} joins the company. How will you update your data-structure?

⚠ Add {F, 10} @ record[6]

😞 But record[6] does not exists. The array record is of size 5 only.

Now what will you do?

(1) Make a new array newrecord of size 6

(2) Copy the five records from record to newrecord.

(3) Add {F, 10} @ newrecord[6]

(1) Make a new array newrecord of size 6

(2) Copy the five records from record to newrecord.

(3) Add {F, 10} @ newrecord[6]

Q: What is the running time of this procedure?

(1) Make a new array newrecord of size 6

(2) Copy the five records from record to newrecord.

(3) Add {F, 10} @ newrecord[6]

Q: What is the running time of this procedure?

A: O(n) if there are n records in record array.

When an employee leaves

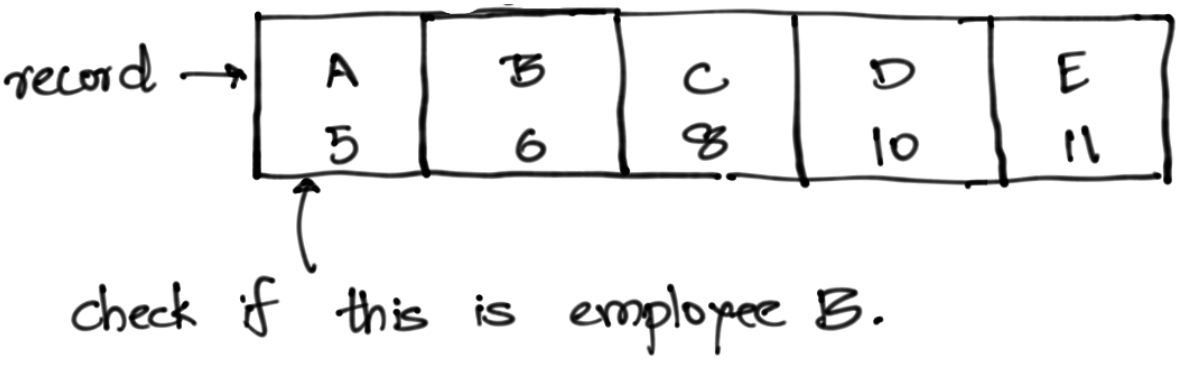
Suppose employee B leaves, so we have to remove the record of employee B.

Q: How would you do that?

When an employee leaves

Suppose employee B leaves, so we have to remove the record of employee B.

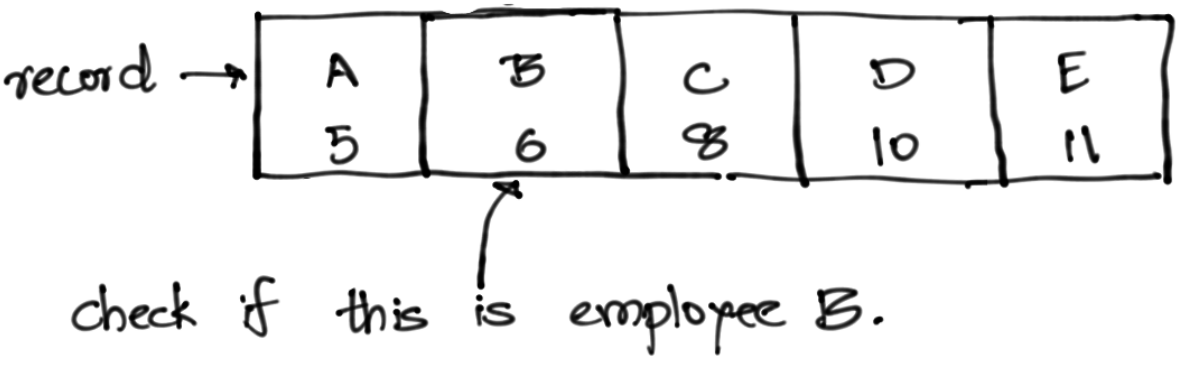
Q: How would you do that?



When an employee leaves

Suppose employee B leaves, so we have to remove the record of employee B.

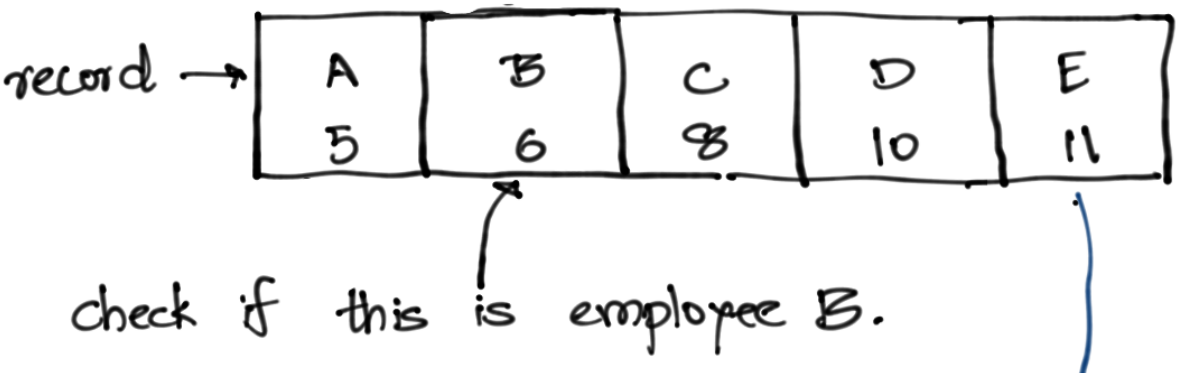
Q: How would you do that?

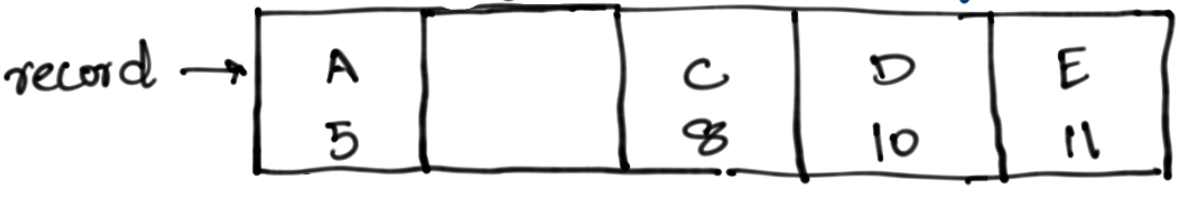


When an employee leaves

Suppose employee B leaves, so we have to remove the record of employee B.

Q: How would you do that?

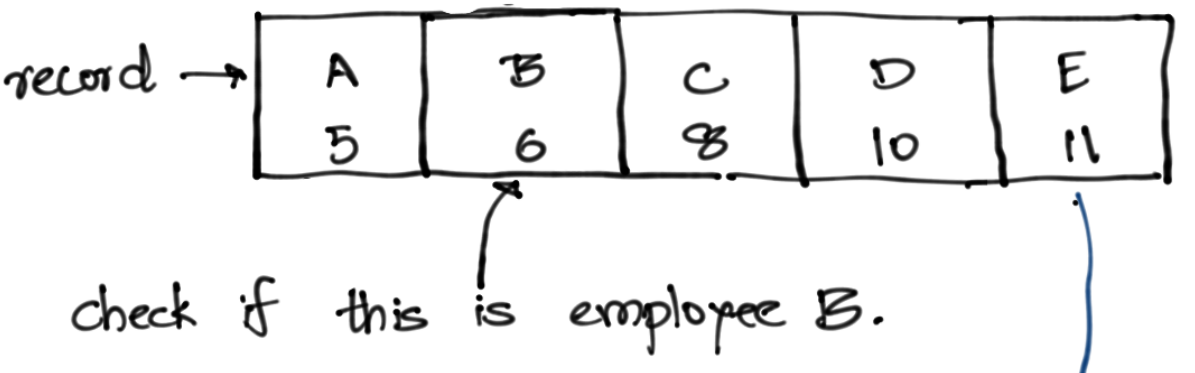


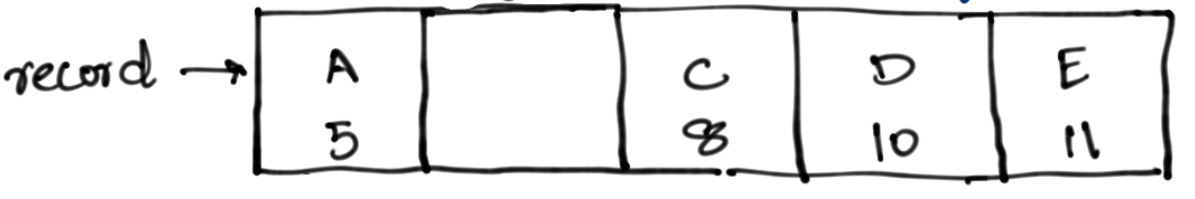


When an employee leaves

Suppose employee B leaves, so we have to remove the record of employee B.

Q: How would you do that?



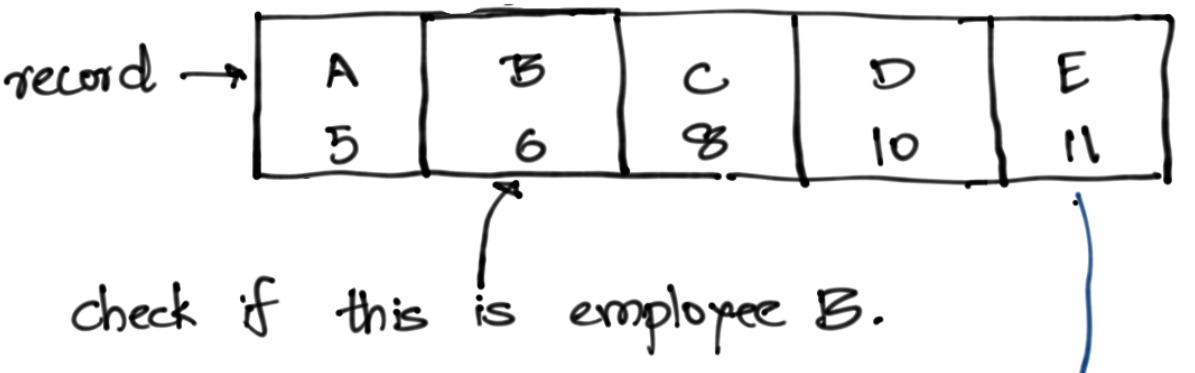


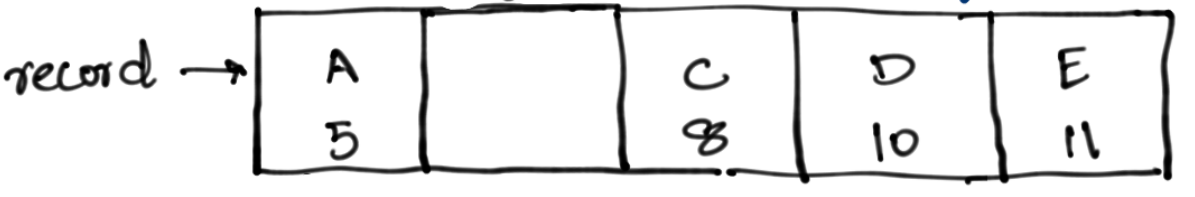
Q: What is the running time of this method?

When an employee leaves

Suppose employee B leaves, so we have to remove the record of employee B.

Q: How would you do that?





Q: What is the running time of this method?

A: O(n) if there are n records.

Q: Is there any other problem with the deletion procedure?

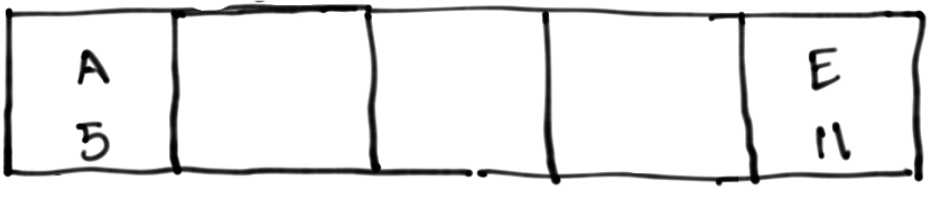
Q: Is there any other problem with the deletion procedure?

A: (1) Wastage of Space: the number of employees may be much lesser than the record array.

Q: Is there any other problem with the deletion procedure?

A: (1) Wastage of Space: the number of employees may be much lesser than the record array.

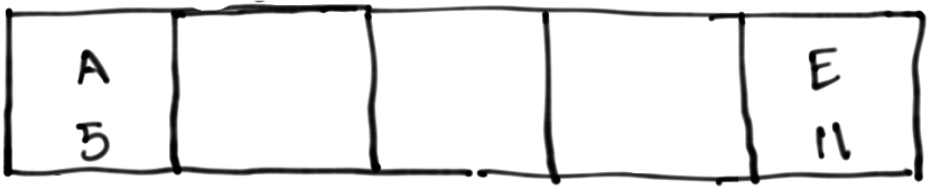
(2) The time taken to search an employee will not be proportional to the # employees.



Q: Is there any other problem with the deletion procedure?

A: (1) Wastage of Space: the number of employees may be much lesser than the record array.

(2) The time taken to search an employee will not be proportional to the # employees.



Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	O(n)

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	O(n)

So, array nearly always give worst case performance.

Q: Where are arrays good ?

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	O(n)

So, array nearly always give worst case performance.

Q: Where are arrays good !

A : Give me the 5th element of record array.

return record[5]

Running Time = ??

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	O(n)

So, array nearly always give worst case performance.

Q: Where are arrays good!

A: Give me the 5th element of record array.

return record[5]

Running Time = O(1).

Performance of Arrays.

Insert	$O(n)$
Deletion	$O(n)$
Search	$O(n)$
Get the k^{th} element	$O(1)$.

Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

There is a simple data-structure which solves this problem.

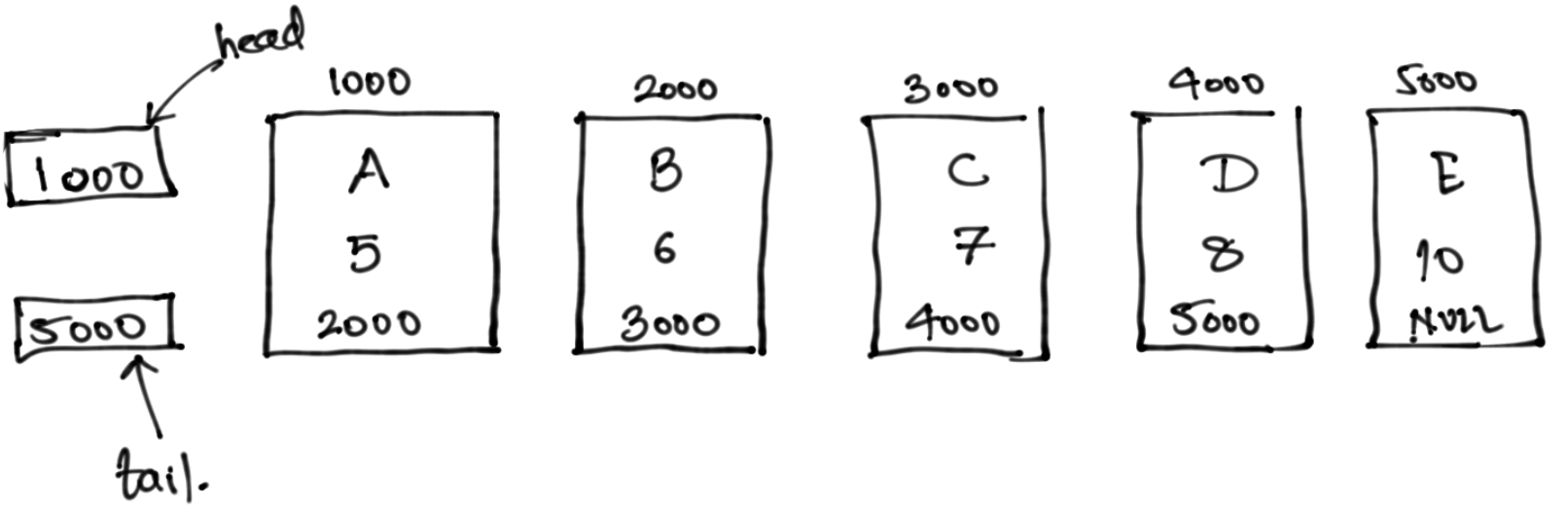
Mimics arrays.

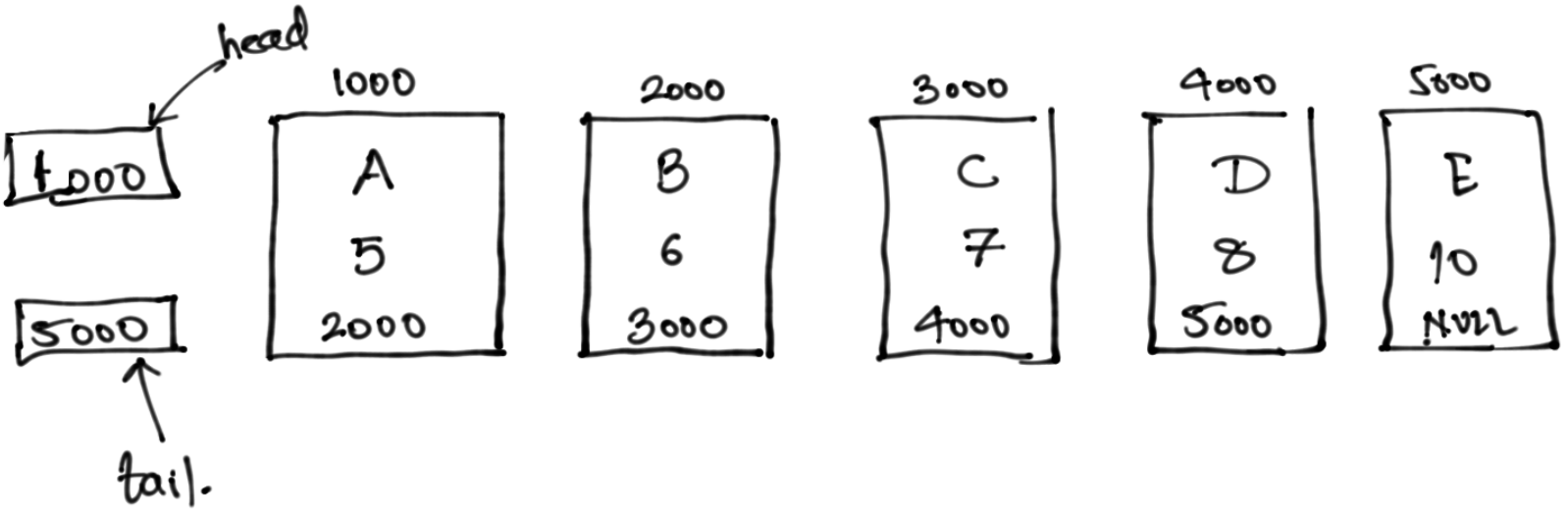
Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

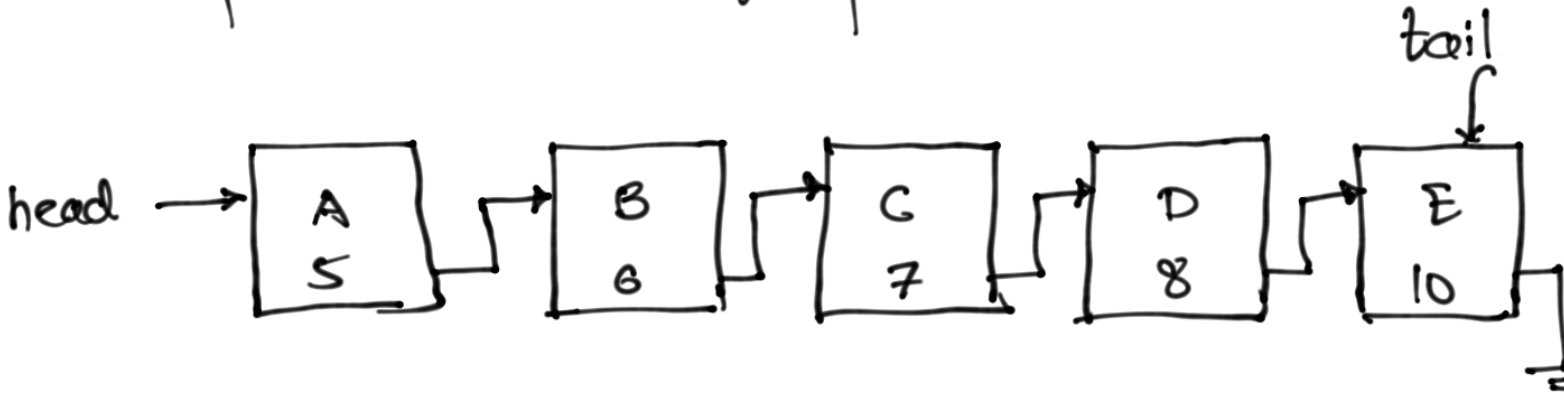
There is a simple data-stoucture which solves this problem.

Mimics arrays.


head
1000
5000
tail.
1000
A
5
2000
2000
B
6
3000
3000
C
7
4000
4000
D
8
5000
5000
E
10
NULL



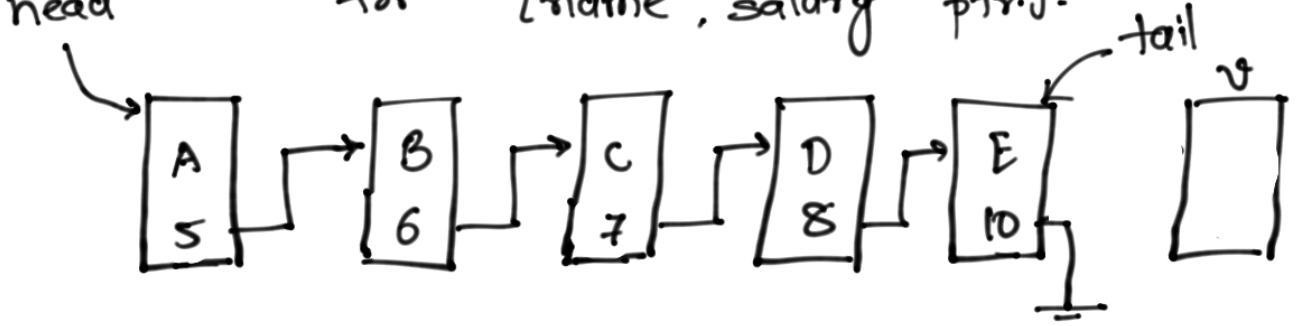
However, we will have a simpler pictorial view of pointers.



Insert {F. 10}.

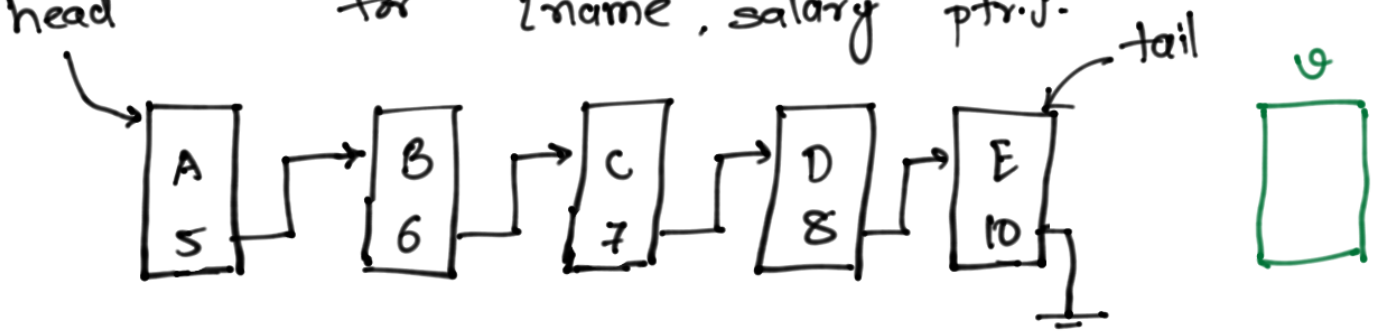
Insert {F, 10}.

① v ← Request the OS to allocate enough space for {name, salary ptr.}.

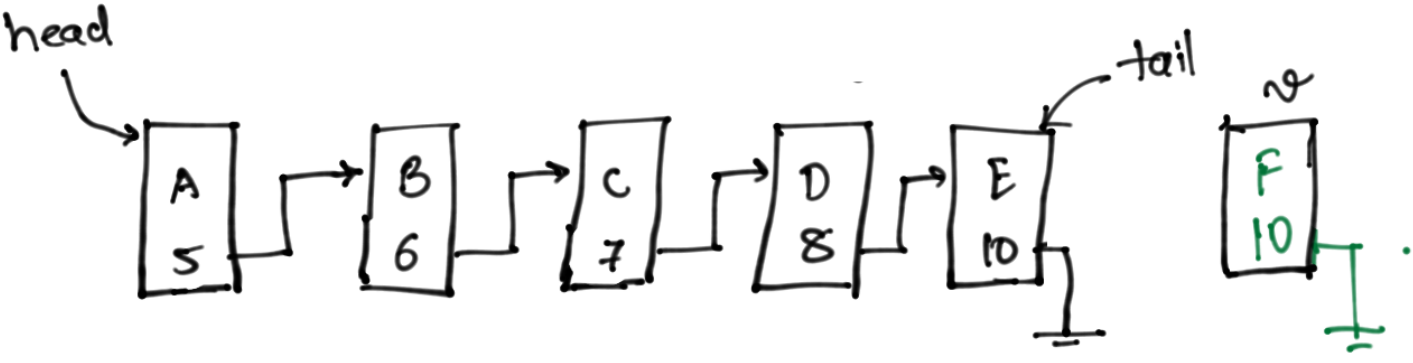


Insert {F, 10}.

① v ← Request the OS to allocate enough space for {name, salary ptr.}.

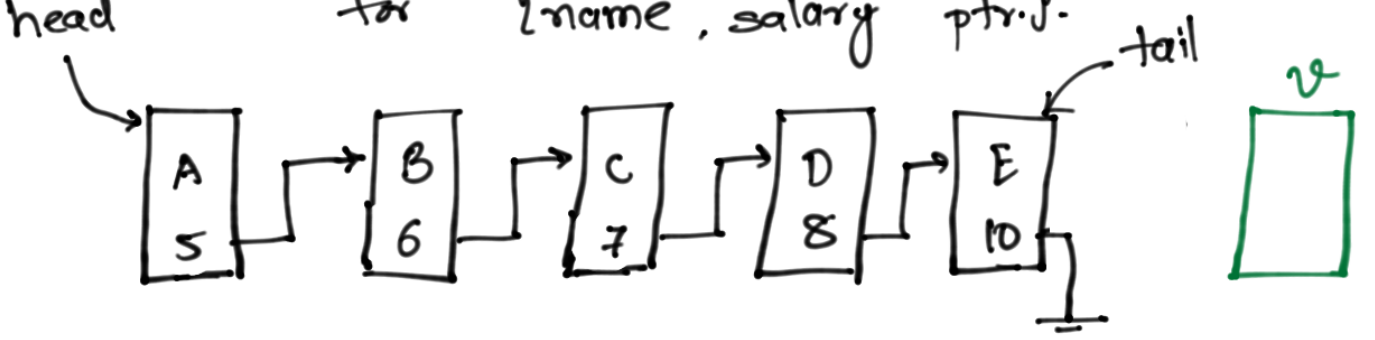


② v.name ← F ;
v.salary ← 10;
v.ptr ← NULL

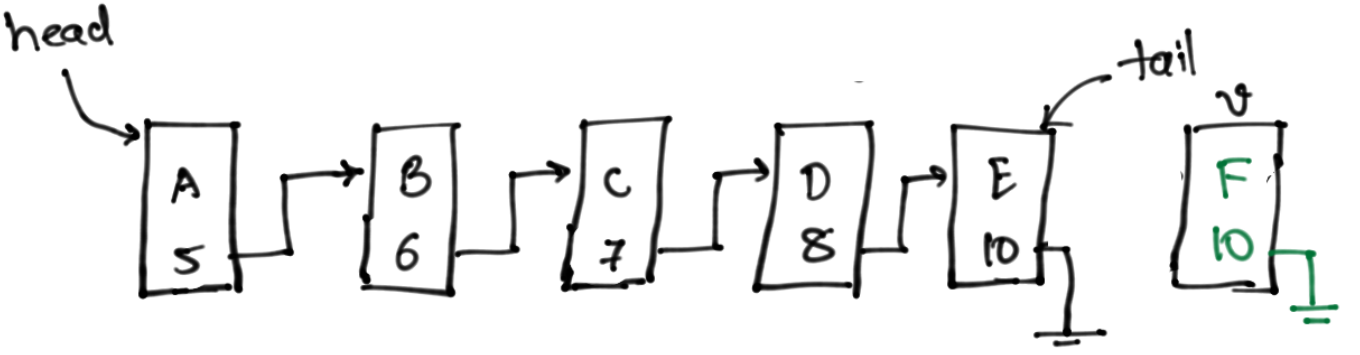


Insert {F, 10}.

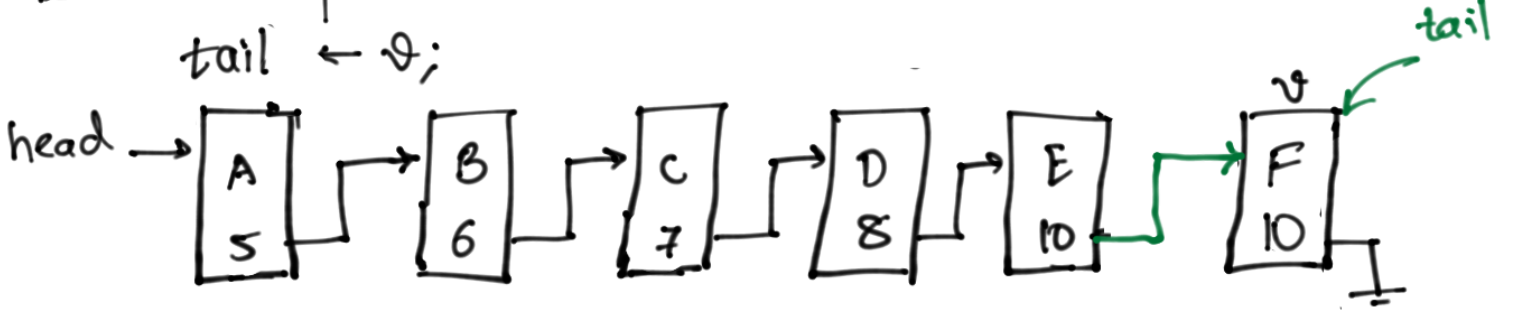
① v ← Request the OS to allocate enough space for {name, salary ptr.}.



② v.name ← F ;
v.salary ← 10;
v.ptr ← NULL



③ tail.ptr ← v ;
tail ← v;



```
Insert (name, salary)
{
    v ← allocate me a new memory location;
    v.name ← name;
    v.salary ← salary;
    v.ptr ← NULL;

    if (head is NULL).
    {
```

```
Insert (name, salary)
{
    v ← allocate me a new memory location;
    v.name ← name;
    v.salary ← salary;
    v.ptr ← NULL;

    if (head is NULL).      // list is empty
    {
        head ← v;
        tail ← v;
    }
    else
    {
```

```
Insert (name, salary)
{
    v ← allocate me a new memory location;
    v.name ← name;
    v.salary ← salary;
    v.ptr ← NULL;

    if (head is NULL).    // list is empty
    {
        head ← v;
        tail ← v;
    }
    else
    {
        tail.ptr ← v;
        tail ← v;
    }
}
```

Q: What is the running time of this procedure ?

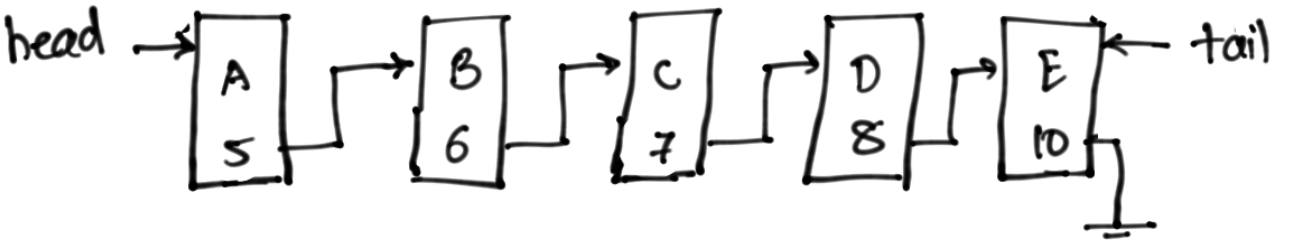
```
Insert (name, salary)
{
    v ← allocate me a new memory location;
    v.name ← name;
    v.salary ← salary;
    v.ptr ← NULL;

    if (head is NULL).    // list is empty
    {
        head ← v;
        tail ← v;
    }
    else
    {
        tail.ptr ← v;
        tail ← v;
    }
}
```

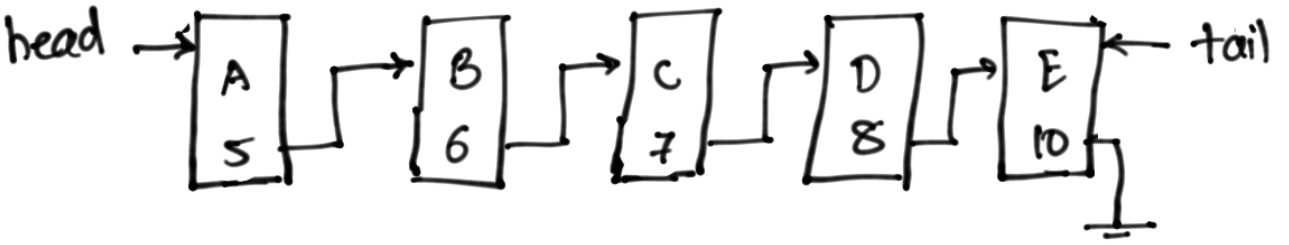
Q: What is the running time of this procedure?

A: O(1).

delete Employee C

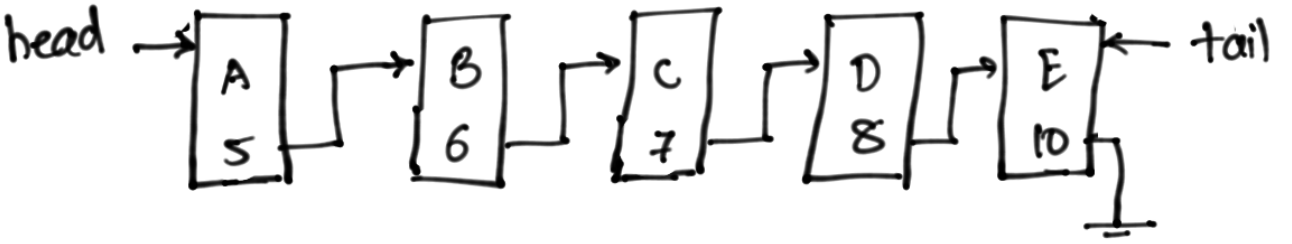


delete Employee C



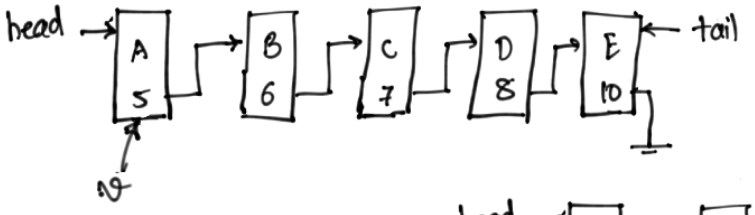
① $v \leftarrow$ head (start with the head).

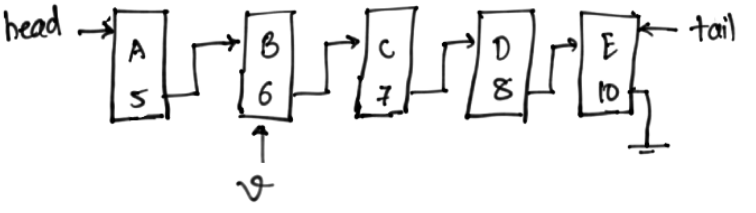
delete Employee C

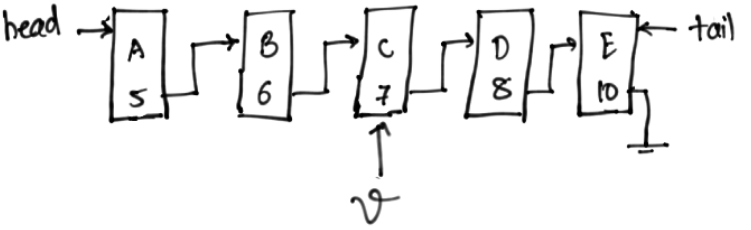


① $v \leftarrow$ head (start with the head).

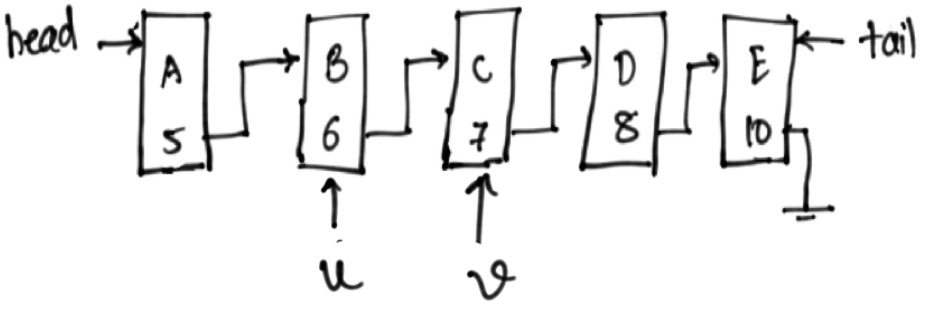
② Move through the list (using pointers) till you hit employee C





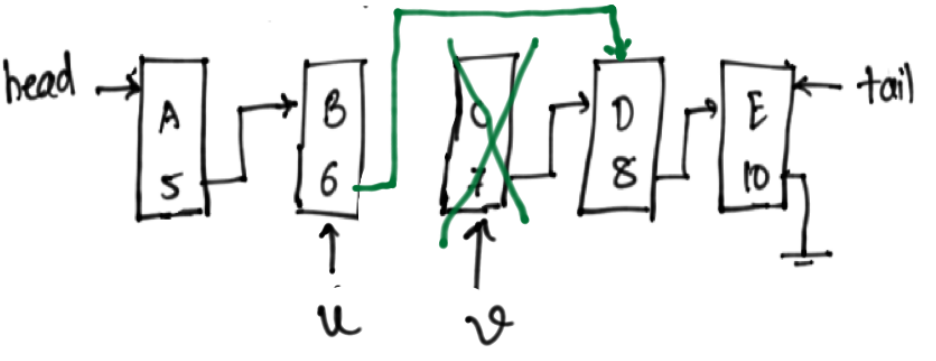


③ Maintain a ptr that follows v, say u.



④ $u.ptr \leftarrow v.ptr$

deallocate the memory associated with record



```
Delete(name)
{
        if ( head is NULL)
            return;
```

```
Delete(name)
{
    if ( head is NULL)
        return;

    if (                    )
    {

    }
    else
    {   v ← head.ptr
        u ← head;
        while ( v is not NULL)
        {
            if (v.name = name)
            {   u.ptr ← v.ptr
                deallocate the memory allocated
                                to record v;
            }
            else
            {   u ← v;
            }   v ← v.ptr
        }
}
```

```
Delete(name)
{
    if ( head is NULL)
        return;

    if ( head.name = name)
    {   v ← head;
        head ← head.ptr;
        deallocate the memory allocated to
                                record v
    }
  else
  {   v ← head.ptr
      u ← head;
      while ( v is not NULL)
      {
          if (v.name = name)
          {  u.ptr ← v.ptr
             deallocate the memory allocated
                          to record v;
          }
          else
          {    u← v;
          }    v ← v.ptr
      }
}
```

Q: What is the running time of Delete(·)

Q: What is the running time of Delete(·)

A $O(n)$

Performance of Linked List

Insert	O(1)
Delete	O(n)
Search	?!

Performance of Linked List

Insert	O(1)
Delete	O(n)
Search	O(n)

Performance of Linked List

Insert	O(1)
Delete	O(n)
Search	O(n)

But the most important thing:
Holy Grail for data-structure

> The space taken by your data-structure should be proportional to the number of current employees in the company.

Q: When is linked list worse than arrays?

Performance of Linked List

Insert	O(1)
Delete	O(n)
Search	O(n)

But the most important thing:
Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

Q: When is linked list worse than arrays?

Query: Get me k^{th} record.

Time taken by linked list = ?!

Performance of Linked List

Insert	O(1)
Delete	O(n)
Search	O(n)

But the most important thing:
Holy Grail for data-structure

The space taken by your data-structure should be proportional to the number of current employees in the company.

Q: When is linked list worse than arrays?

Query: Get me k^{th} record.

Time taken by linked list = O(k)

Performance of Arrays.

Insert	O(n)
Deletion	O(n)
Search	O(n)
Get the k^{th} element	O(1).

Performance of Linked List

Insert	O(1)
Deletion	O(n)
Search	O(n)
Get the k^{th} element	O(k)