

Introduction to Algorithms and Data Structures

Lesson 7: Data Structure (1) Data structures for search algorithms

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

<http://www.jaist.ac.jp/~uehara>

Algorithm and Data structure

- Algorithm: The method of solving a problem
- Data structure:
 - Format of data and intermediate results of computation
 - It contributes efficiency of algorithmsExample: Array, linked list, stack, queue, priority queue, tree structure

We introduce some basic ones using search problem

Array: Easy to access

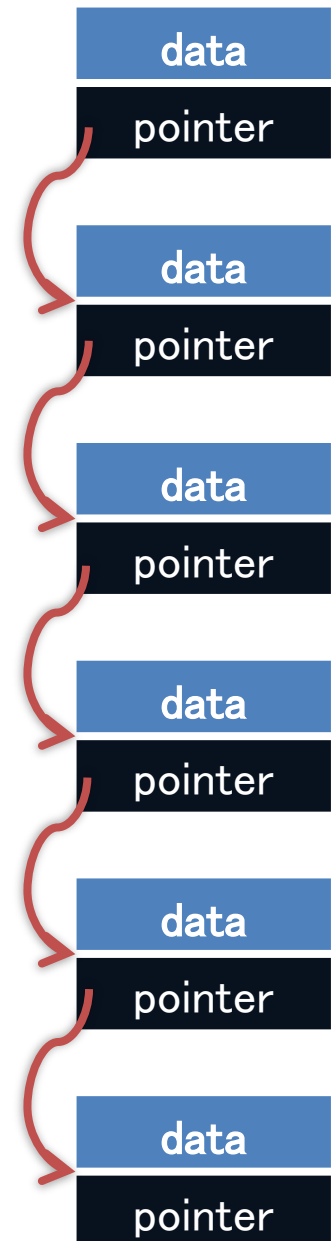
- By **random access** property (in RAM model), it takes a constant time to access any data when we specify its *index*.
 - cf. There are some data structures that only allow to access from its top
 - ➔ It takes $O(i)$ time for access to the i -th element
e.g., linked list
- It can be accessed in order of indices; that is, it has **sequential access** property.
 - cf. There are some data structures that lack of this property
e.g., tree structure

Linked list

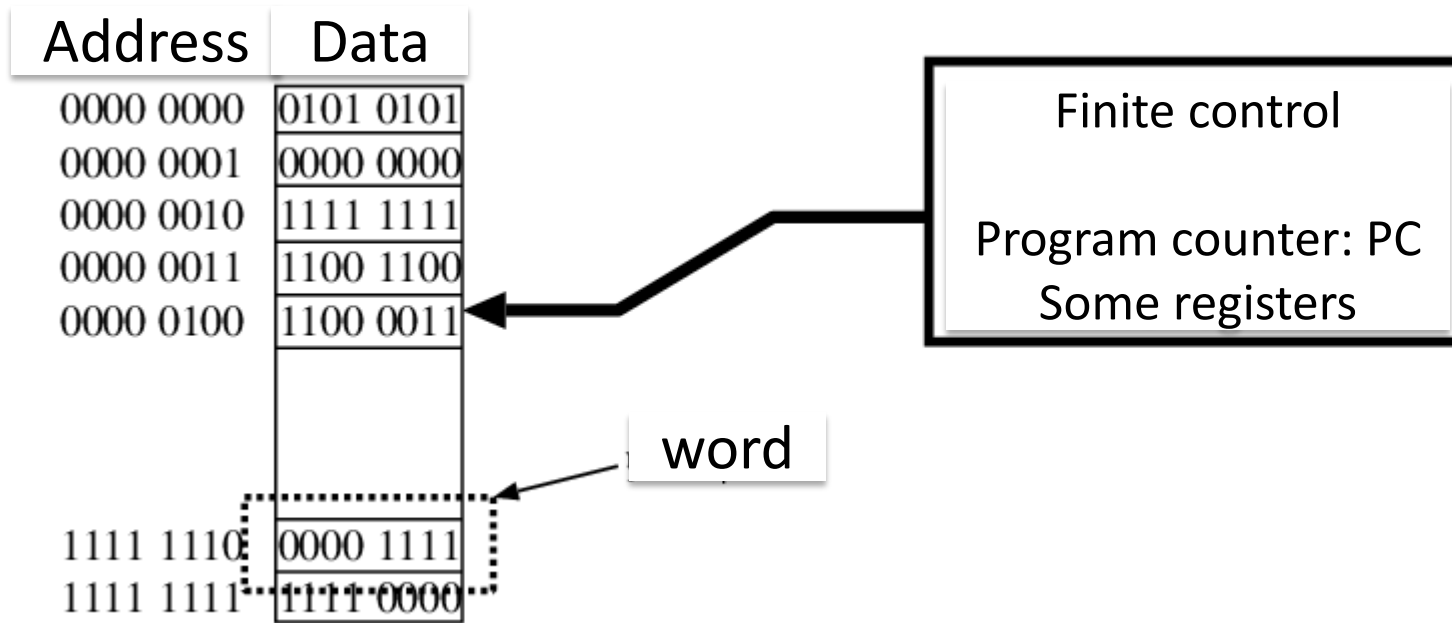
- It indicates “next/backward” elements explicitly
- Set of records
 - Data: it stores data
 - Pointer: it indicates the next element
- Some variants
 - One-way linked list
 - Two-ways linked list
 - It can represent a tree



Report 3:
What's a
“pointer” in
RAM model?



C.f.: RAM Model

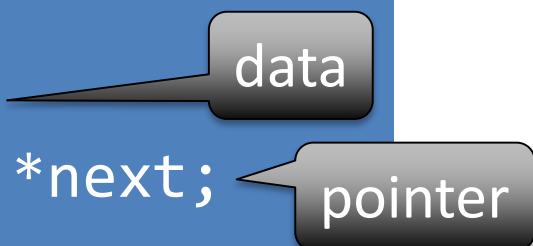


- It consists Memory and CPU (Central Processing Unit)
 - We do not mind Input/Output
- It is essentially the same as your computer
- CPU can access any address randomly (not sequentially) in a unit cycle
- Programming language C is a system that show you this structure implicitly (like arrays and pointers)

One-way linked list

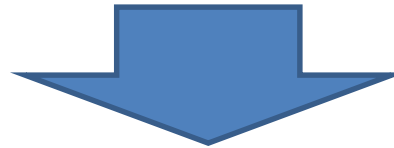
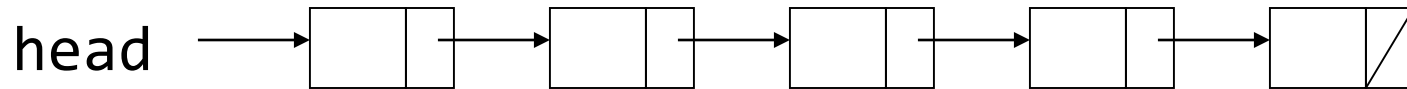
- Sequence of records
 - data: it stores data
 - pointer: it indicates the next record

```
typedef struct{  
    int data;  
    struct list_t *next;  
} list_t;  
list_t *new_r;  
new_r =  
    (list_t *)  
    malloc(sizeof(list_t));
```



Example: Store many data into one-way linked list

- Base:
 - Generate record r in memory
 - Store x in the data area of r
 - Connect r to the list



New record r

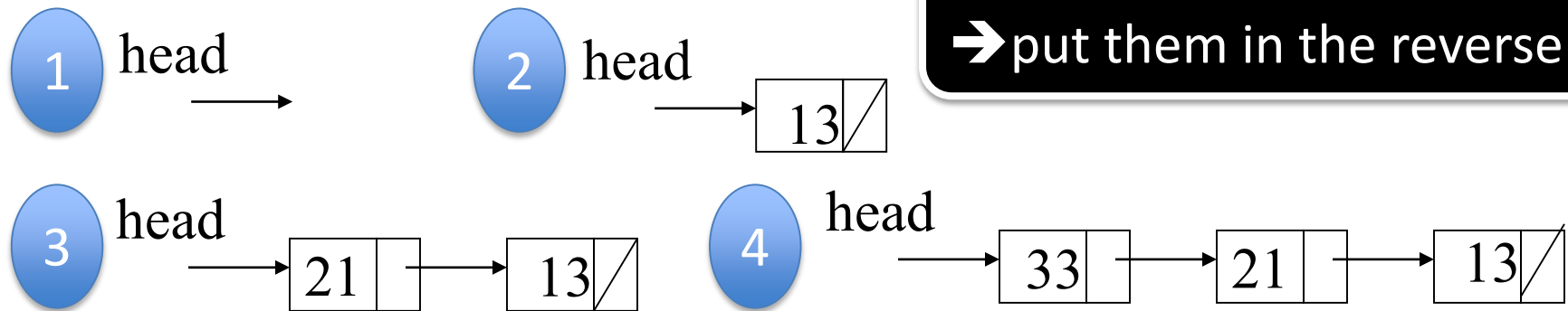


- Connect to the first or last item in the list

Program that adds a new record at the head of the one-way linked list

```
list_t *head, *new_r;  
int x;  
head = NULL;  
while(/*there are new data*/){  
    new_r = (list_t *)  
            malloc(sizeof(list_t));  
    new_r->data = x;  
    new_r->next = head; head = new_r;  
}
```

New record is added to the top
→ put them in the reverse order



Program that adds a new record at the tail of the one-way linked list

```
list_t *head, *new_r, *tail;
int x = /*some value*/;
new_r =(list_t *)
    malloc(sizeof(list_t));
new_r->data = x;
new_r->next = NULL; head = new_r; tail = new_r;
while(/*there are data*/){
    x=/* next data */;
    new_r =(list_t *)
        malloc(sizeof(list_t));
    new_r->data = x;
    tail->next = new_r;
    new_r->next = NULL; tail = new_r;
}
```

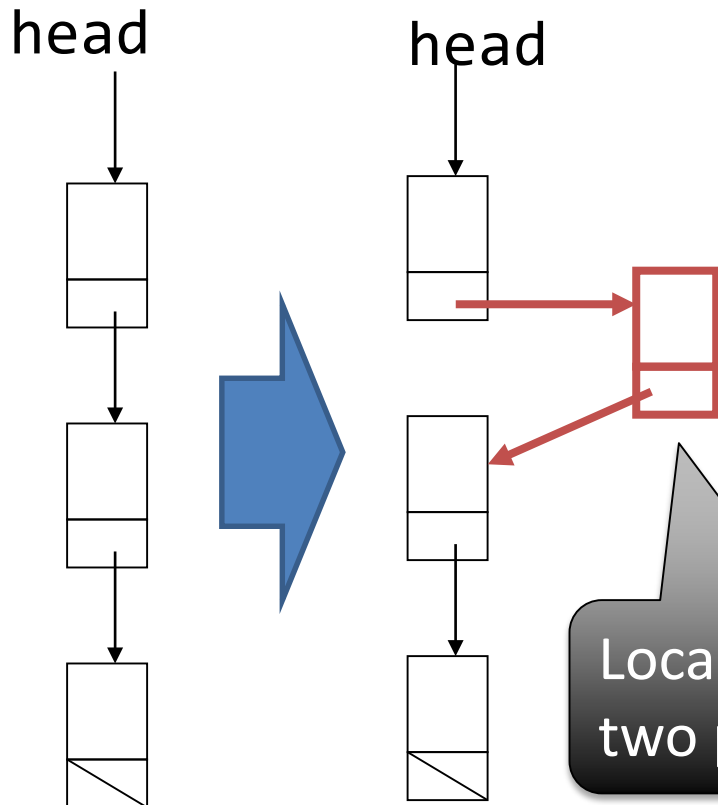
Pointer that indicates the last record

tail is updated to indicate the new record

Advantage of linked list (comparing to array): “Insert” is easy!

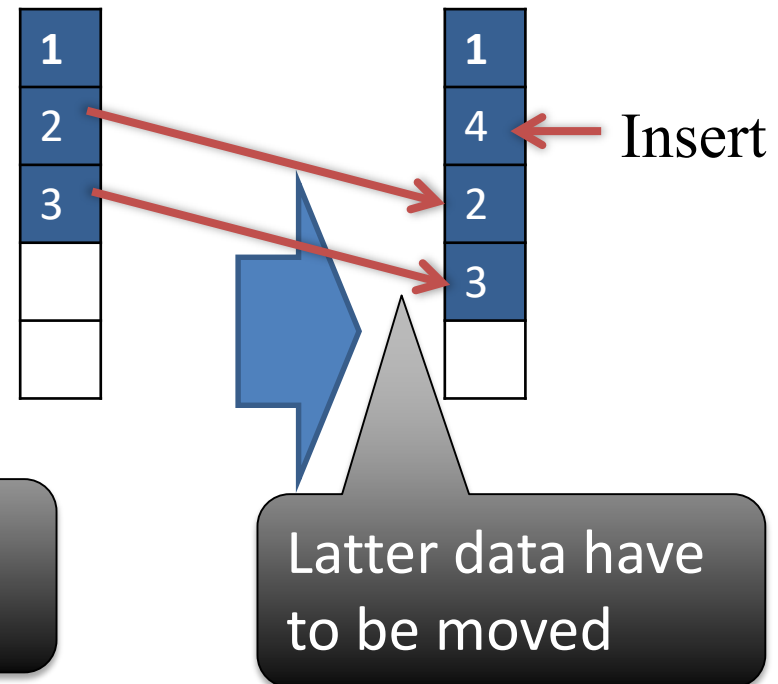
Linked list

- No need to move data



Array

- We have to move (many) data

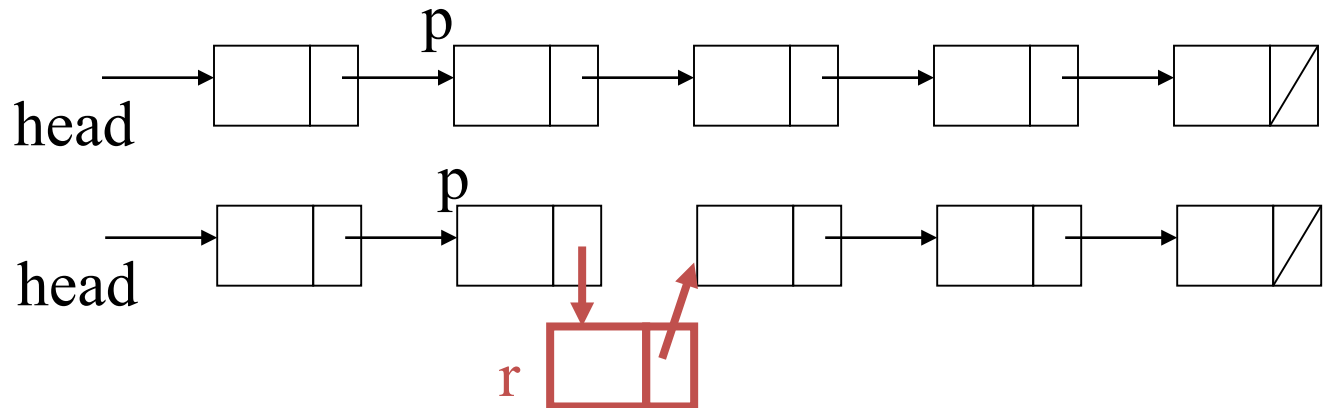


One-way linked list: Insertion of data

(Insert a new item after p)

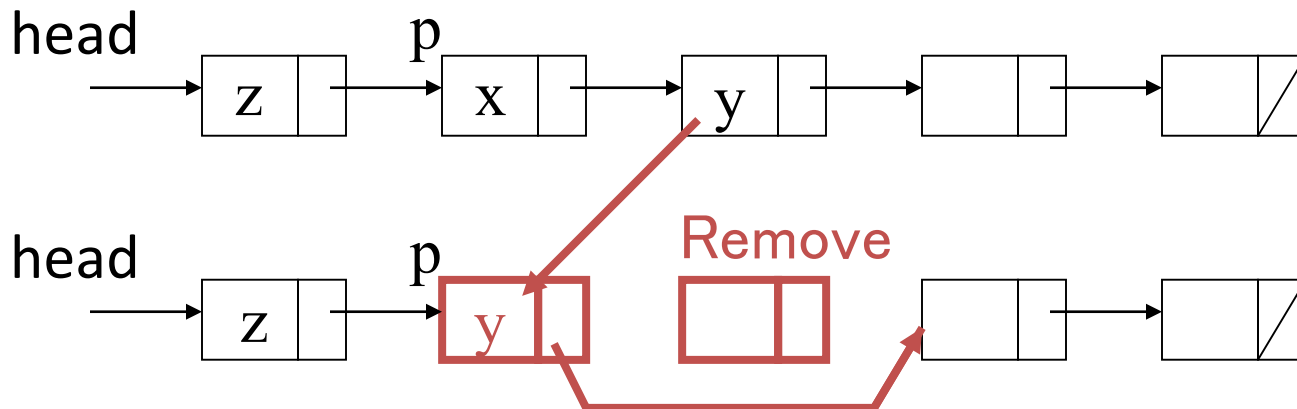
- Make a record r that has x as data
- Copy “p->next” of p to the pointer of r
- Update the pointer of p to indicate r

```
new_r = (list_t *)  
        malloc(  
            sizeof(list_t)  
        );  
new_r->data = x;  
new_r->next = p->next;  
p->next = new_r
```



One-way linked list: Deletion of data

- Remove a record that pointer p indicates
 - $p \rightarrow \text{data} = p \rightarrow \text{next} \rightarrow \text{data}$
 - $p \rightarrow \text{next} = p \rightarrow \text{next} \rightarrow \text{next}$



Note: Removal of x is bit tricky. When algorithm checks x, it already forgets the address of z.