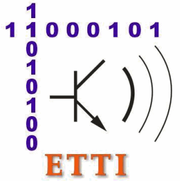
UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI



Liste simplu înlănțuite

Cazuri particulare

Cuprins

Partea I – Noțiuni teoretice

1. Set (mulțimi)
2. Map (funcție asociativă)
3. Priority queue (coadă cu priorități)
4. Insertion sort

Partea II – Aplicații Laborator

Rezolvare aplicații Moodle

1. Set

Listă de date simplu înlănțuită care implementează conceptul de **mulțimi**, alături de toate operațiile specifice. Într-o mulțime, fiecare element este **unic** (nu se repetă). Funcții implementate, pe lângă operațiile uzuale de la liste simplu înlănțuite:

- `contains`: verifică dacă un element face parte din mulțime. Implicit, necesită implementarea logicii de comparare a două elemente;
- `intersect`: intersecția a două mulțimi;
- `union`: reuniunea a două mulțimi;
- `subtract`: scăderea a două mulțimi.

```
typedef struct node{
    <tip_data> key;
struct node *next;
} node_t;
```

2. Map

Listă de date simplu înlănțuită care implementează conceptul de **funcție asociativă**. Fiecare nod reprezintă o pereche de tipul `(key, value)`. Fiecărei chei îi corespunde o singură valoare, iar cheile sunt unice în întreaga listă. Funcții implementate, pe lângă operațiile uzuale de la liste simplu înlănțuite:

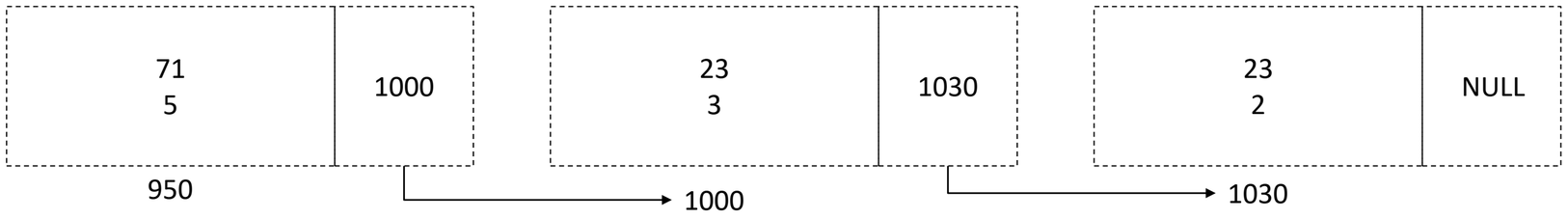
- `contains`: verifică dacă o cheie face parte din mulțime. Implicit, necesită implementarea logicii de comparare a două elemente;
- `get`: pentru o cheie dată, returnează valoarea asociată.
- `set`: pentru o cheie dată, suprascrie valoarea asociată.
- `keys`: returnează mulțimea tuturor cheilor din listă.
- `values`: returnează mulțimea tuturor valorilor din listă.

```
typedef struct node{
    <tip_data> key;
    <tip_data> value;
    struct node *next;
} node_t;
```

3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

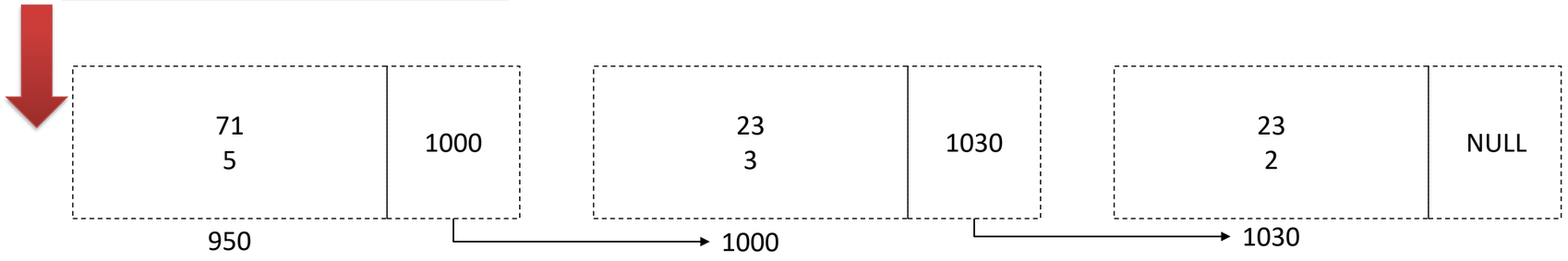


3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

```
enqueue(14, 9);
```

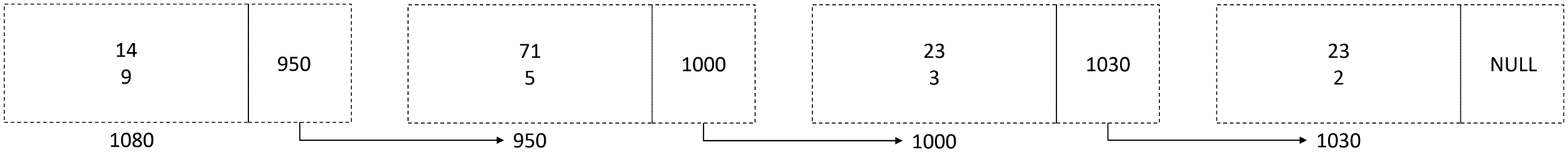


3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

```
enqueue(14, 9);
```

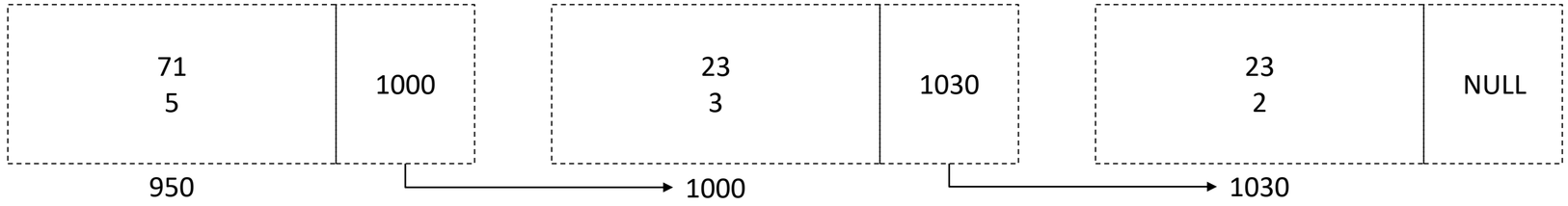


3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

```
enqueue(14, 9);
dequeue();
```

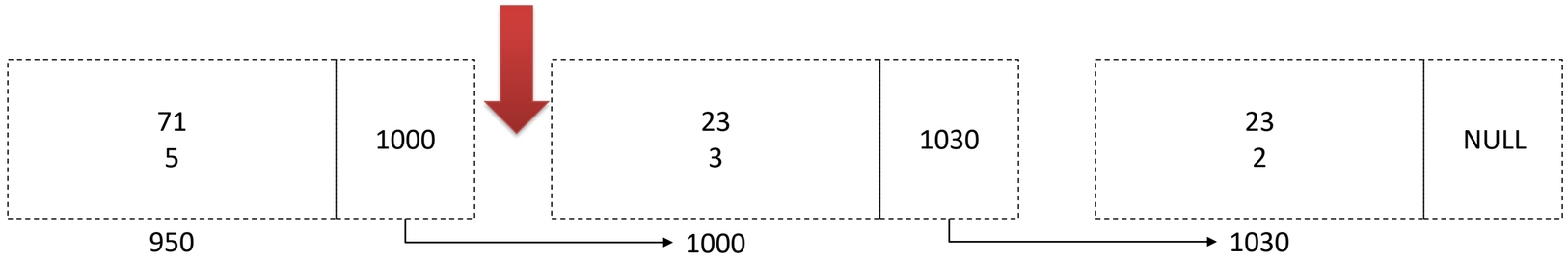


3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

```
enqueue(14, 9);
dequeue();
enqueue(53, 4);
```

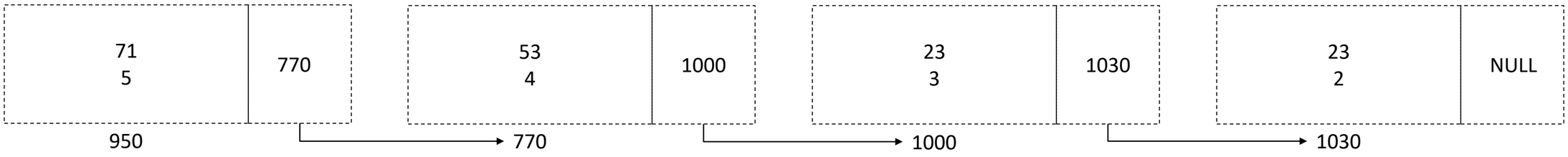


3. Priority queue

Listă de date simplu înlănțuită care implementează conceptul de **coadă cu priorități**. Fiecare nod conține, pe lângă datele utile, o valoare reprezentând prioritatea sa. Elementele de prioritate mare vor fi procesate primele (în mod implicit, dacă nu este specificată o altă regulă). Spre deosebire de coada normală, funcția de adăugare de elemente (enqueue/push) inserează elementul nou pe o poziție astfel încât să se mențină ordinea priorităților în interiorul cozii.

```
typedef struct node{
    <tip_data> data;
    int priority;
    struct node *next;
} node_t;
```

```
enqueue(14, 9);
dequeue();
enqueue(53, 4);
```



4. Insertion sort

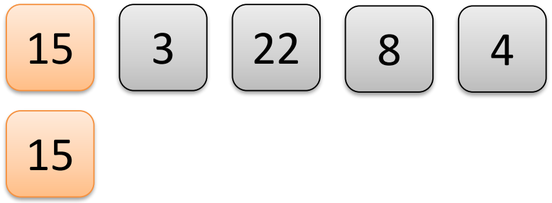
Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.

4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

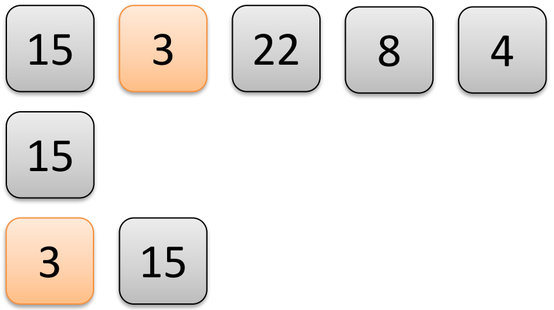
1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.



4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

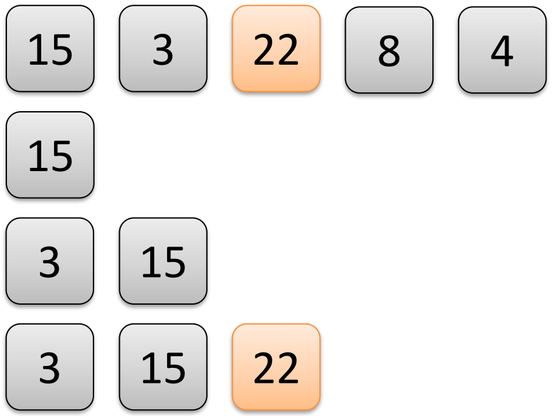
1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.



4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

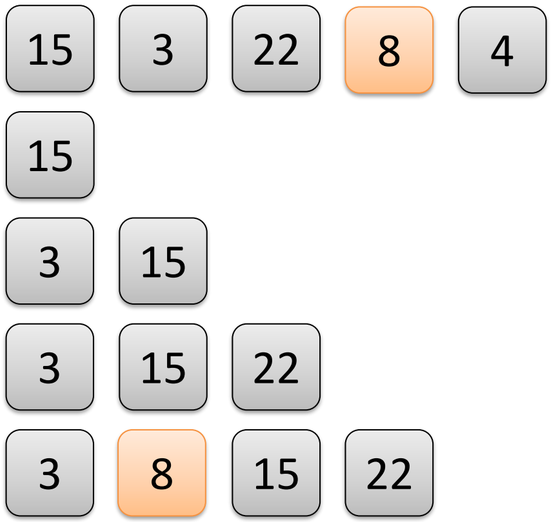
1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.



4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

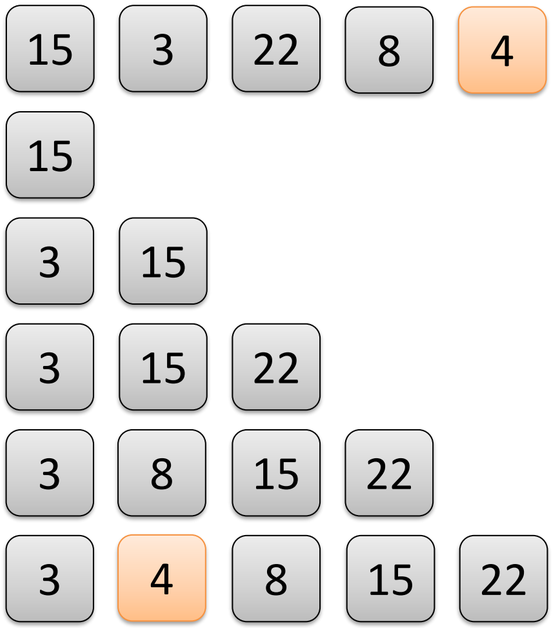
1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.



4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.



4. Insertion sort

Coada cu priorități implementează nativ o formă de sortare a elementelor în funcție de prioritate. Similar, structurile de tip Set și Map pot implementa o formă sortată a cheilor sale. Un algoritm simplu de sortare este Insertion Sort. Se urmăresc pașii de mai jos:

1. Se creează o nouă listă ce va conține elementele sortate.
2. Lista inițială este parcursă element cu element.
3. Fiecare element preluat din lista originală este introdus pe poziția sortată în lista finală.