# ЛЕКЦІЯ 2

## 2.1. Структури даних

**Означення.** Опис складних об'єктів засобами більш простих типів даних, які безпосередньо представляються у машині, називається *структурами даних*.

**Означення.** *Списком* (довжини n) називається впорядкована послідовність елементів $a_1$, $a_2$, ..., $a_n$. Список розміру 0 називається *порожнім*.

**Означення.** Список можна реалізувати або за допомогою масиву або за допомогою зв'язування його елементів вказівниками (*зв'язаний список*). У зв'язаному списку елементи лінійно впорядковані, їх порядок визначається вказівниками, що входять у склад елементів списку.

**Означення.** Елемент *двостороннього зв'язаного списку* містить три поля: ключ та два вказівника – наступний та попередній. В *односторонньому зв'язаному списку* відсутнє поле 'попередній'. У *впорядкованому списку* елементи розташовані в порядку зростання ключів на відміну від *невпорядкованого списку*.

**Означення.***Стеки* та *черги* – це динамічні множини (або спеціальні типи списків), в яких елемент що додається, визначається структурою множини. Стек працює за принципом "останній прийшов – перший пішов (LIFO)", а черга – за принципом "перший прийшов – перший пішов (FIFO)".

*Приклади структур даних.*

Нехай $S$ - деяка множина, представлена структурою даних, і нехай $u$ - довільний елемент деякої множини, підмножиною якого $S$. Наведемо основні операції, які зустрічаються при роботі із множинами:

1. ПРИНАЛЕЖНІСТЬ ($u$,$S$). Вірно, що $u \in S$?(Відповідь типу ДА/НІ).
2. ВСТАВИТИ ($u$,$S$).Включить $u$ у $S$.
3. ВИЛУЧИТИ ($u$,$S$). Вилучити $u$ із $S$.

   Припустимо, що { $S_1$ , $S_2$ ,..., $S_k$ }- набір множин (які попарно не перетинаються). На цьому наборі корисні такі операції:
4. ЗНАЙТИ ($u$). Знайти таке $j$, що $u \in S_j$.
5. ОБ'ЄДНАТИ ($S_i$ ,$S_j$; $S_k$). Сформувати об'єднання $S_i$ і $S_j$ і назвати його $S_k$.

   Якщо універсальна множина повністю упорядкована, то дуже важливі такі операції:
6. *MIN(S).* Знайти найменший елемент $S$.
7. РОЗЧЕПИТИ ($u$,$S$). Розділити $S$ на $\{S_1, S_2\}$ такі ,що $S_1=\{v: v \in S$ і $v \le u\}$, а $S_2 = S\text{-}S_1$.
8. ЗЧЕПИТИ ($S_1$ ,$S_2$ ). Припустимо, що для будь-яких $u' \in S_1$ і $u'' \in S_2$ маємо $u' \le u''$; необхідно сформувати множину $S = S_1 \cup S_2$.

| Структури даних | Допустимі операції |
|---|---|
| Словник | НАЛЕЖНІСТЬ, ВСТАВИТИ, ВИЛУЧИТИ |
| Пріоритетна черга | *MIN*, ВСТАВИТИ, ВИЛУЧИТИ |
| Зчеплена черга | ВСТАВИТИ, ВИЛУЧИТИ, РОЗЧЕПИТИ, ЗЧЕПИТИ |

## 2.2. Дерево відрізків.

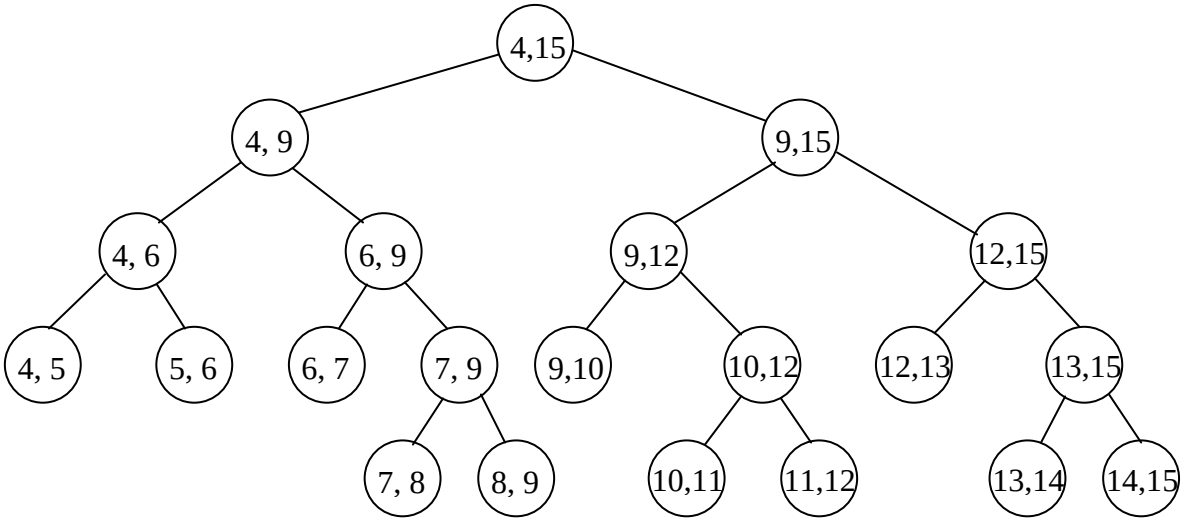
**Означення.**.*Дерево відрізків* - це структура даних, створена для роботи з такими інтервалами на числовій осі, кінці яких належать *фіксованій* множині із $N$ абсцис (ці абсциси можна нормалізувати, заміняючи кожну із них її порядковим номером при обході їх зліва на право). Можна вважати ці абсциси цілими числами на інтервалі [1,$N$].

*Дерево відрізків* - це двійкове дерево з коренем. Для заданих цілих чисел $l$ і r таких, що $l< r$, дерево відрізків T$(l,r)$ будується рекурсивно таким чином :

Складається із кореня $v$ *з параметрами* $B[v]=l$ та $E[v]=r$; (B – Begin, E - End)

1. Якщо $r-l>1$, то воно складається із лівого піддерева $T(l,\lfloor (B[v]+E[v])/2 \rfloor)$ та правого піддерева $T(\lfloor (B[v]+E[v])/2 \rfloor ,r)$. ( Корені піддерев природно позначити через ЛСИН[$v$] та ПСИН[$v$] відповідно).
2. Параметри $B[v]$ та $E[v]$ позначають інтервал $[B[v], E[v]] \subseteq [l, r]$ , зв'язаний з вузлом $v$.

Інтервали, що належать множині {[B[v], E[v]]: v - вузол T($l$, $r$)}, називаються *стандартними інтервалами* дерева T($l$, $r$). Стандартні інтервали, які належать листам T($l$, $r$), називаються *елементарними інтервалами*.T($l$, $r$) збалансоване і має глибину $\lceil \log_2(r - l) \rceil$.




Мал.. 1.1. Дерево відрізків T(4, 15)

## ***Алгоритм вставки інтервалу [b,e]***

Фрагментація інтервалу [*b,e*] повністю визначається операцією, яка вставляє [*b,e*] в дерево відрізків *T*, і зверненням ВСТАВИТЬ (*b,e;* корінь (*T*)) до наступної процедури:

**Procedure** ВСТАВИТЬ (b, e; v)

**begin if (** b ≤ B[v] ) **and** ( E[v] ≤ e) **then** призначить [b, e] вузлу v

**else begin if** ( b < ⌊( B[v]+E[v] )/2⌋ ) **then**

ВСТАВИТЬ ( b, e; ЛСИН[v] );

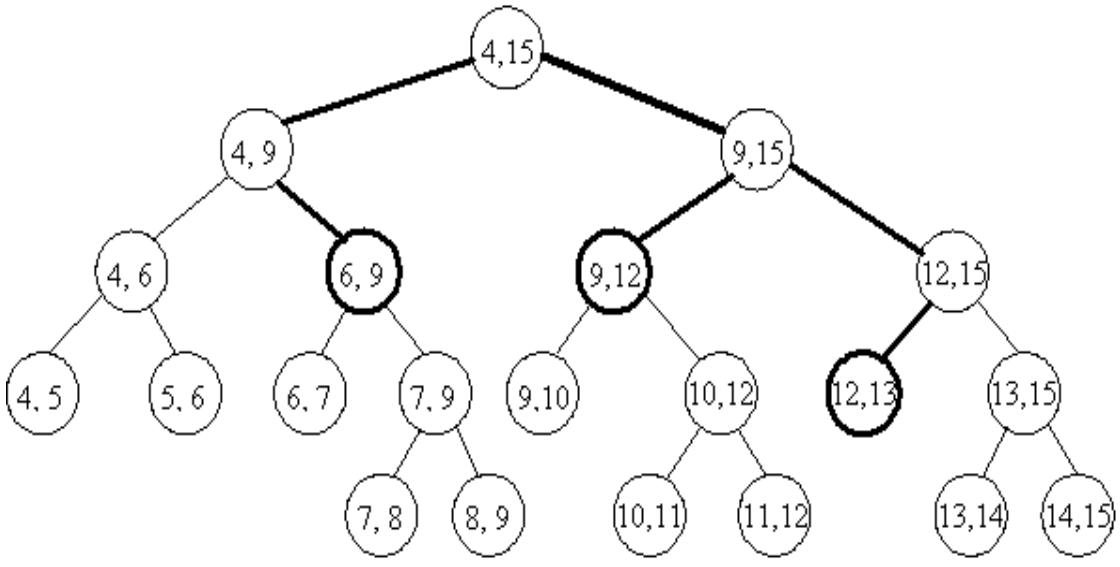
**if** (⌊( B[v] + E[v])/2⌋ < e ) **then**

ВСТАВИТЬ ( b, e; ПСИН [v] )

**end**

**end**

Дія ВСТАВИТЬ ( b, e; корінь ( *T* ) ) відповідає "маршруту" в *T* , який має загальну структуру (мал.1.2):




Мал. 1.2. Вставка інтервала [6, 13] в T(4, 15). Вузли віднесення виділені жирним.

1) початковий шлях $\boldsymbol{P}_{\text{поч.}}$ від кореня до вузла v*, який наз. *розгалуженням*, із якого виходять два шляхи - $\boldsymbol{P}_{\text{л}}$ і $\boldsymbol{P}_{\text{П}}$ .

2) інтервал, що вставляється відноситься або повністю до розгалуження, або до усіх правих синів шляху $\boldsymbol{P}_{\text{л}}$ та $\boldsymbol{P}_{\text{П}}$ ; при цьому визначається фрагментація [*b,e*] (*вузли віднесення)*.

**Procedure** ВИЛУЧИТЬ (b, e; v)

**begin if (** b ≤ B[v] ) **and** ( E[v] ≤ e) **then** C[v]:=C[v]-1.

**else begin if** ( b < ⌊( B[v]+E[v] )/2⌋ ) **then**

ВИЛУЧИТЬ ( b, e; ЛСИН[v] );

**if** (⌊( B[v] + E[v])/2⌋ < e ) **then**

ВИЛУЧИТЬ ( b, e; ПСИН [v] )

**end**

**end**

## 2.3. Реберний список з подвійними зв'язками ( РСПЗ) .

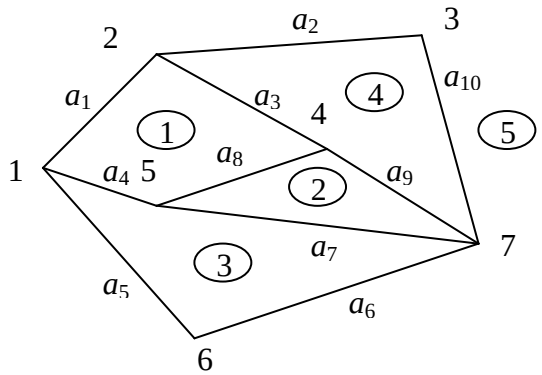
Нехай V = {$v_1$, $v_2$, …, $v_N$}, а E = {$e_1$, $e_2$,…,$e_M$ }. Головна компонента РСПЗ для планарного графа (V, E) це *реберний вузол* , який містить чотири інформаційні поля ( V1, V2, F1, F2 ) і два поля вказівників (P1 і P2):

Поле V1 - містить початок ребра, поле V2- містить його кінець; так ребро отримує умовну орієнтацію.

Поля F1 і F2 містять імена граней, які лежать відповідно праворуч і ліворуч від ребра, орієнтованого від V1 до V2.

Вказівник P1 ( відповідно P2 ) задає реберний вузол, який містить перше ребро , яке зустрічається слідом за ребром (V1, V2), при повороті від нього проти часової стрілки навколо V1 (відповідно V2) .

Імена граней і вершин можуть бути задані цілими числами. На малюнку 1.3 показано фрагмент РСПЗ.



| | $V_1$ | $V_2$ | $F_1$ | $F_2$ | $P_1$ | $P_2$ |
|---|---|---|---|---|---|---|
| $a_1$ | 1 | 2 | 5 | 1 | 5 | 3 |
| $a_2$ | 2 | 3 | 5 | 4 | 1 | 10 |
| $a_3$ | 2 | 4 | 4 | 1 | 2 | 8 |
| $a_4$ | 1 | 5 | 1 | 3 | 1 | 7 |
| $a_5$ | 1 | 6 | 3 | 5 | 4 | 6 |
| $a_6$ | 6 | 7 | 3 | 5 | 5 | 10 |
| $a_7$ | 5 | 7 | 2 | 3 | 8 | 6 |
| $a_8$ | 5 | 4 | 1 | 2 | 4 | 9 |
| $a_9$ | 4 | 7 | 4 | 2 | 3 | 7 |
| $a_{10}$ | 3 | 7 | 5 | 4 | 2 | 9 |

Мал. 1.3 РСПЗ

Якщо граф має N вершин та F граней, то введемо два масиви HV[1:N] та HF[1:F], які містять номер ребра, що виходить з відповідної вершини (обмежує грань). Процедура ***Fill*** заповнює ці масиви переглядаючи V1 та F1 за час O(N).

```
Fill
for i ← 1 to N do
begin
    if (HV[V1[i]] = 0) then HV[V1[i]] ← i;
    if (HV[V2[i]] = 0) then HV[V2[i]] ← i;
    if (HF[F1[i]] = 0) then HF[F1[i]] ← i;
    if (HF[F2[i]] = 0) then HF[F2[i]] ← i;
end;
```

Після виконання процедури ***Fill*** для наведеного прикладу масиви *HV* та *HF* приймуть наступні значення:

*HV* = [1,1,2,3,4,5,6];

*HF* = [1,7,4,2,1].

Процедура **vertex** ( *i* ) будує послідовність ребер, інцидентних $v_i$ як послідовність адрес, занесених в масив An.

```
Vertex ( j )
i ← 1;
a ← HV[j];
a0 ← a;
An[i] ← a;
i ← i + 1;
if (V1[a] = j) then a ← P1[a]
   else a ← P2[a];
while (a ≠ a0) do
```

```
begin
   An[i] ← a; i ← i + 1;
   if (V1[a] = j) then a ← P1[a]
      else a ← P2[a];
end;
```

Час роботи процедури ***Vertex*** ( *j* ) пропорційний числу ребер, інцидентних $v_j$. Аналогічно можна створити процедуру ***Facet*** ( *j* ), за допомогою якої можна отримати послідовність ребер, які обмежують грань $f_j$, замінивши в попередній процедурі HV та V1 на HF та F1.

Процедура ***Vertex*** ( *j* ) перелічує ребра в порядку обходу навколо вершини проти годинникової стрілки, а ***Facet*** ( *j* ) перелічує їх в порядку обходу за годинниковою стрілкою навколо грані.