Дърво

Дърво е абстрактен тип данна с възел, наречен корен на дървото, който е свързан с едно или повече дървета, наречени поддървета, или не е свързан с нито едно поддърво.

Клон е връзка между корена и всяко негово поддърво. Всеки клон може да води до нови разклонения, но не допуска друг път до корена.

Родител е възел, който е свързан към един или повече възела, наречени негови наследници.

Лист е възел, който няма наследници.

Всеки възел в дървото се адресира чрез уникален път от възли.

Дърветата са йерархични по природа.

Двоично дърво е дърво, в което всеки възел има най-много два наследника – ляв наследник и десен наследник.

Има корен, ляво поддърво (ЛПД) и дясно поддърво (ДПД).

Възел – има данни и връзки към ляво и дясно поддърво.

Данни – ключ и други възможни компоненти.

Двоично дърво за претърсване (подредено дърво) е дърво, на което ключът на левия наследник на всеки възел е по-малък от ключа на самия възел и ключът на десния наследник е по-голям от този на възела. Дървото е сортирано според ключовете на възлите си.

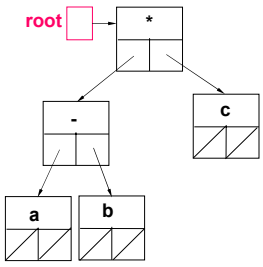
Приложение

– декодиране на морзов код;

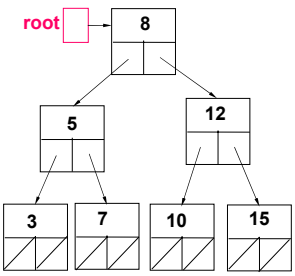
– интерактивни компютърни игри за отгатване.

Дефиниране

```
typedef <тип> KEY;
struct data                     // данни
{       KEY key;                // ключ
        <... други възможни компоненти;>
};
typedef struct data DATA;
struct node                     // възел
{       DATA data;              // данни
        struct node *left;      // връзка към ЛПД
        struct node *right;     // връзка към ДПД
};
typedef struct node NODE;
typedef NODE *LINK;
struct tree                     // дърво
{       LINK root;              // корен на дърво
};
typedef struct tree TREE;
```



Представяне на израза (a-b)*c чрез двоично дърво



Двоично дърво за претърсване

Характеристики

1. **Добавяне на възел – намира се мястото на новия възел и след това става ляв или десен наследник на намерения възел.**
2. **Изтриване на възел – първо се намира мястото на възела; ако е лист, се прекъсва връзката му към дървото; ако има двама наследници, възелът се замества с най-левия възел от дясното поддърво или с най-десния възел от лявото поддърво.**

3. **Преминаване през дърво:**
   - **префиксно („пред“) – коренът се посещава преди поддърветата.**
   - **суфиксно („след“) – коренът се посещава след поддърветата).**
   - **инфиксно („във“) – ЛПД, корен, ДПД.**

Операции

1. **Създаване на празно дърво**
2. **Включване на възел в дърво**
3. **Изтриване на възел от дърво**
4. **Отпечатване на елементите на дърво**

```
#define DUPLICATE -1            // дублиране на възел
```

1. Създаване на празно дърво

Алгоритъм

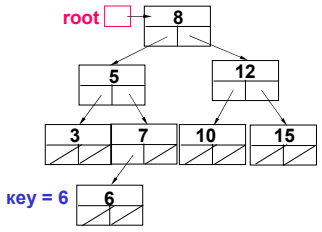
корен ← ПРАЗНО


```
void create (TREE *t)
{
  t->root = NULL;           // празно дърво
}
```

2. Включване на възел в дърво

Алгоритъм

ако дървото е празно
** отдели памет за нов възел**
** насочи корен към новия възел**
в противен случай
** намери място в дървото, започвайки от корен**



отделяне на памет за нов възел
отдели памет за нов възел
ако отделянето на памет не е успешно
** неуспешна операция**
в противен случай
** запази данните в новия възел**
** връзка към ЛПД ← ПРАЗНО**
** връзка към ДПД ← ПРАЗНО**
** насочи намерената връзка към новия възел**
** успешна операция**

намиране на място в дървото
ако ключът на данните на новия възел е равен на ключа на данните на текущия възел
  възелът вече е в дървото
в противен случай
  ако ключът е по-малък от текущия ключ
    ако няма ЛПД
      отдели памет за нов възел
      насочи левия наследник към новия възел
      успешна операция
    в противен случай
      намери място в ЛПД
  в противен случай
    ако ключът е по-голям от текущия ключ
      ако няма ДПД
        отдели памет за нов възел
        насочи десния наследник към новия възел
        успешна операция
      в противен случай
        намери място в ДПД

```
тип КЛЮЧ: ЦЯЛО;
тип данни = (телефонен номер: КЛЮЧ;
             име: НИЗ;
             адрес: НИЗ)
```

```
        дърво          данни за нов възел
int add (TREE *t, DATA item)          // Включва възел в дърво
{
  if (t->root == NULL)                // ако дървото е празно
    return allocate (&t->root, item); // отделя памет за корен
  else
    return locate (t->root, item);    // намира място в дървото
}
```

```
      намерена връзка          данни за новия възел
int allocate (LINK *node, DATA item)   // Отделя памет за възел
{
  LINK p;
  p = (LINK)malloc(sizeof(NODE));      // отделя памет
  if (p == NULL)
  {
    printf("Няма достатъчно памет!\n");
    return FAILURE;
  }
```

```
  else
  {
    p->data.key = item.key;          // запазва телефонния номер
    p->data.name = (char *)malloc(strlen(item.name)+1);
    if (p->data.name == NULL)
    {
      printf("Няма достатъчно памет!\n");
      return FAILURE;
    }
    else
    {
      strcpy(p->data.name, item.name);          // запазва името
      p->data.address = (char *)malloc(strlen(item.address)+1);
      if (p->data.address == NULL)
      {
        printf("Няма достатъчно памет!\n");
        return FAILURE;
      }
```

```
      else
      {
        strcpy(p->data.address, item.address); // запазва адреса
        p->left = NULL;          // връзка към ЛПД ← ПРАЗНО
        p->right = NULL;         // връзка към ДПД ← ПРАЗНО
        *node = p;               // насочва връзката към възела
        return SUCCESS;
      }
    }
  }
}
```

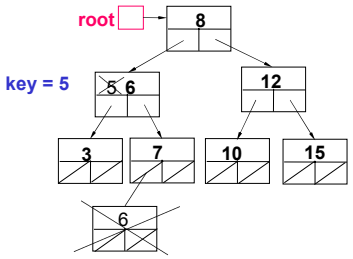
текуща връзка — данни за новия възел

```
int locate (LINK p, DATA item)      // Намира място в дърво
{  if (item.key == p->data.key)     // в дървото има такъв ключ
     return DUPLICATE;
   else if (item.key < p->data.key) // ако ключът е по-малък
   {  if (p->left == NULL)                    // няма ЛПД
        return allocate (&p->left, item);
      else if (locate(p->left, item)==DUPLICATE) // търсене в ЛПД
        return DUPLICATE;
   }
   else                             // ако ключът е по-голям
   {  if (p->right == NULL)                   // няма ДПД
        return allocate (&p->right, item);
      else                                    // търсене в ДПД
        if (locate(p->right, item) == DUPLICATE)
          return DUPLICATE;
   }
   return SUCCESS;
}
```

3. Изтриване на възел от дърво

Алгоритъм

ако дървото е празно
** неуспешна операция**
в противен случай
** намери мястото на възела, започвайки от корен**



намиране на мястото на възела
ако ключът на данните на възела е равен на ключа на данните на текущия възел
** изтрий елемента, посочен от текущия възел**
** успешна операция**
в противен случай
** ако ключът е по-малък от текущия ключ**
**   ако няма ЛПД**
**   неуспешна операция**
**   в противен случай**
**   търси в ЛПД**
** в противен случай**
**   ако ключът е по-голям от текущия ключ**
**   ако няма ДПД**
**   неуспешна операция**
**   в противен случай**
**   търси в ДПД**

изтриване на елемент
ако възелът съществува
** ако възелът е лист**
**   изтрий листа**
** в противен случай**
**   ако възелът няма ЛПД**
**   изтрий десния лист**
**   в противен случай**
**   ако възелът няма ДПД**
**   изтрий левия лист**
**   в противен случай**
**   намери най-левия възел в ДПД**
**   замени данните във възела с данните на най-левия възел**

намиране на най-левия възел в поддърво
ако поддървото не е ПРАЗНО
** ако левият наследник на поддървото е ПРАЗНО**
**   запази данните на най-левия възел**
**   изтрий най-левия възел**
** в противен случай**
**   търси най-левия възел**

дърво — данни на възел за изтриване

```
int del (TREE *t, DATA item)        // Изтрива възел от дърво
{
   if (t->root == NULL)             // ако дървото е празно
     return FAILURE;                // неуспешна операция
   else
     return locdel (&t->root, item); // намира мястото на възела
                                     // и го изтрива
}
```

връзка за изтриване — данни на възел за изтриване

```
int locdel (LINK *p, DATA item)     // Намира мястото на възела
{
   if (item.key == (*p)->data.key)  // ако възелът е намерен
   {
     deleteitem (p);                // изтрива възела
     return SUCCESS;
   }
   else if (item.key < (*p)->data.key)    // ако ключът е по-малък
     if ((*p)->left == NULL ||            // ако няма ЛПД
         locdel (&((*p)->left),item) == FAILURE)
       return FAILURE;                    // неуспешна операция
   else if (item.key > (*p)->data.key)    // ако ключът е по-голям
     if ((*p)->right == NULL ||           // ако няма ДПД
         locdel (&((*p)->right),item) == FAILURE)
       return FAILURE;                    // неуспешна операция
}
```

```
                                  връзка за изтриване
void deleteitem (LINK *q)          // Изтрива елемент
{  LINK d;                         // временна връзка
   DATA replace;
   if (q != NULL)                  // ако връзката сочи към възел
   {
      if ((*q)->left == NULL && (*q)->right == NULL)     // ако е лист
      {
         free(*q);                 // освобождава паметта за възела
         *q = NULL;                // изтрива връзката към листа
      }
      else
         if ((*q)->left == NULL)       // ако няма ЛПД
         {                             // изтрива десния лист
            d = *q;       // насочва временна връзка към възела
            *q = (*q)->right;    // свързва връзката към ДПД
            free(d);             // освобождава паметта за възела
         }
```

```
         else    // Проверка за липса на десен наследник
            if ((*q)->right == NULL)   // ако няма ДПД
            {                          // изтрива левия лист
               d = *q;      // насочва временна връзка към възела
               *q = (*q)->left;    // свързва връзката към ЛПД
               free(d);            // освобождава паметта за възела
            }
            else                   // Има два наследника
            {
               // намира най-левия възел в ДПД
               leftmost (&((*q)->right), &replace);
               // заменя данните във възела с данните на най-
               // левия възел в ДПД
               (*q)->data = replace;
            }
   }
}
```

```
връзка към най-левия                          данни на най-левия
възел                                         възел
// Намира най-левия възел в поддърво
void leftmost (LINK *node, DATA *item)
{
   LINK d;                             // временна връзка
   if (*node != NULL)                  // ако поддървото не е празно
   {
      if ((*node)->left == NULL)       // ако няма ляв наследник
      {                                // намерил е най-ляв възел
         *item = (*node)->data;        // запазва данните му
         d = *node;          // насочва временна връзка към възела
         *node = (*node)->right;       // свързва връзката към ДПД
         free (d);                     // освобождава паметта
      }
      else                             // търси най-левия възел
         leftmost (&((*node)->left), item);
   }
}
```

4. Отпечатване на елементите на дърво

Алгоритъм

ако дървото не е ПРАЗНО
** премини през дървото инфиксно, започвайки от корен**

инфиксно преминаване през дърво
ако ЛПД не е ПРАЗНО
** премини през ЛПД**
отпечатай данните на текущия възел
ако ДПД не е ПРАЗНО
** премини през ДПД**

```
void print (TREE t)                // Отпечатва дърво
{  if (t.root != NULL)                  дърво
      inorder (t.root, 0);
}                                       връзка към възел
void inorder (LINK p, int l)  // Инфиксно преминава през дърво
{  if (p->left != NULL)                 отстъп
      inorder (p->left, l+1);
   printnode (p, l);
   if (p->right != NULL)
      inorder (p->right, l+1);
}
void printnode (LINK p, int l)         // Печат на възел
{  int i;
   for (i=1; i<=l; i++)
      printf("   ");
   printf ("%d, %s, %s\n", p->data.key, p->data.name,
           p->data.address);
}
```

Задача: Реализирайте абстрактния тип данни таблица чрез двоично дърво. Таблицата съдържа данни за телефонен номер (ключ), име и адрес.

1. **Напишете функция** createtable()**, която създава таблицата като двоично дърво. (Използвайте функцията** add()**.)**
2. **Отпечатайте таблицата, като използвате функцията** print()**.**

Алгоритъм

докато не е въведен край на файл повтаряй
** въведи данни за елемент: телефонен номер, име, адрес**
** включи елемента в таблицата**
отпечатай елементите на таблицата