▲ Vyberte vhodnou datovou strukturu, kterou byste použili k řešení/simulaci dané úlohy:

1. Kontrola párovosti různých typů závorek v textu.
2. Obsluha I/O požadavků na PC.
3. Zpracování tiskových úloh na tiskárně.
4. Vyhodnocení volání rekurzivních procedur.
5. Evidence účastníků přijímacích zkoušek.
6. Vyhodnocování aritmetických výrazů zapsaných postfixově.
7. Vyřizování požadavků přetíženou linkou technické podpory.

▲ Mějme zásobník definovaný následujícím způsobem:

```
module Stack ( newStack , isEmpty , push , pop ) where

newtype Stack a = S [a]

newStack :: Stack a
newStack = S []

isEmpty :: Stack a -> Bool
isEmpty (S []) = True
isEmpty _      = False

push :: a -> Stack a -> Stack a
push x (S s)   = S (x:s)

pop :: Stack a -> (a,Stack a)
pop (S (x:s)) = (x,S s)
pop _         = error "Stack underflow"
```

Popište postupné přidávání prvků 2, 3, 7, 1 do prázdného seznamu a následné odebrání dvou prvků, společně s jednotlivými „medzivýsledky“.

▲ Navrhněte implementaci dvou zásobníků v poli délky n tak, aby došlo k přetečení obou zásobníků pouze v případě, že počet prvků v obou zásobnících dohromady je větší než n.

▲ Jakou časovou složitost má přidání a odebrání prvku v implementaci z předchozí úlohy?

▲ Mějme cyklickou frontu definovanou nad polem `Q.prvek` pevné délky `n`. Atribut `Q.tail` ukazuje za konec fronty a `Q.head` na čelo fronty. Funkce `Enqueue` má na starosti zařazení prvku do fronty, funkce `Dequeue` jeden prvek z fronty vypustí.

```
1  Queue = record                        |   function Dequeue(Q)
2    prvek: array[1..n] of Integer       |   begin
3    head: Integer                       |     x := Q.prvek[Q.head]
4    tail: Integer                       |     if Q.head = n then
5  end                                   |       Q.head := 1
6                                        |     else
7  function Enqueue(Q,x)                 |       Q.head := Q.head + 1
8  begin                                 |     return x
9    Q.prvek[Q.tail] := x                |   end
10   if Q.tail = n then                  |
11     Q.tail := 1                       |
12   else                                |
13     Q.tail := Q.tail + 1              |
14 end                                   |
```

Jak vypadá inicializace takové cyklické fronty?

Znázorněte jednotlivé stavy fronty během volání následujících funkcí:
`Enqueue(Q,4)`, `Enqueue(Q,7)`, `Enqueue(Q,11)`, `Dequeue(Q)`,
`Enqueue(Q,21)`, `Dequeue(Q)`, `Dequeue(Q)`.

Co se stane, když provedete operaci Dequeue(Q) na čerstvě inicializované frontě Q?

▲ Navrhněte, jak implementovat frontu pomocí dvou zásobníků.

▲ Určete časovou složitost operací `Enqueue(In,Out,x)` a `Dequeue(In,Out)` s frontou definovanou v předchozí úloze pro vstupní sekvenci délky n.

▲ Definice: Seznam (`List`) je dynamická datová struktura, jejíž prvky tvoří posloupnost. Na rozdíl od pole, kde je pořadí prvků určeno pevnými indexy, pořadí v seznamu je dáno vazbami mezi sousedními prvky. V následujících funkcích je zadefinován obousměrně zřetězený spojový seznam a operace nad ním, tj. seznam, kde každý prvek "vidí" (ukazuje na) svého předchůdce i následníka.

▲ Navrhněte proceduru k obousměrnému seznamu, která vrátí prvek seznamu, který je vůči danému prvku `x` relativně posunutý o `k` pozic, kde `k` je celé číslo.

▲ Navrhněte implementaci operací `Insert(S,x)`, `Delete(S,x)` a `Search(S,x)` ve slovníku, který je realizován jednosměrně zřetězeným seznamem. Určete asymptotickou časovou složitost vámi navržených operací.

▲ Navrhněte nerekurzivní proceduru, která obrátí pořadí prvků v jednosměrně zřetězeném seznamu o n prvcích v čase $\Theta(n)$, a to tak, aby procedura využívala pouze konstantní množství paměti nad rámec paměti potřebné k uchování seznamu. Použijte operace seznamu z předcházejícího příkladu, k polím prvků nepřistupujte přímo.

▲ Nakreslete binární strom od uzlu `root`, který je reprezentovaný kódem:

```
root = Node (3,A) a b
a = Node (9,K) c d
b = Node (7,F) Empty e
c = Node (1,D) Empty Empty
d = Node (4,G) Empty Empty
e = Node (6,W) f g
f = Node (10,O) Empty Empty
g = Node (5,I) Empty Empty
```

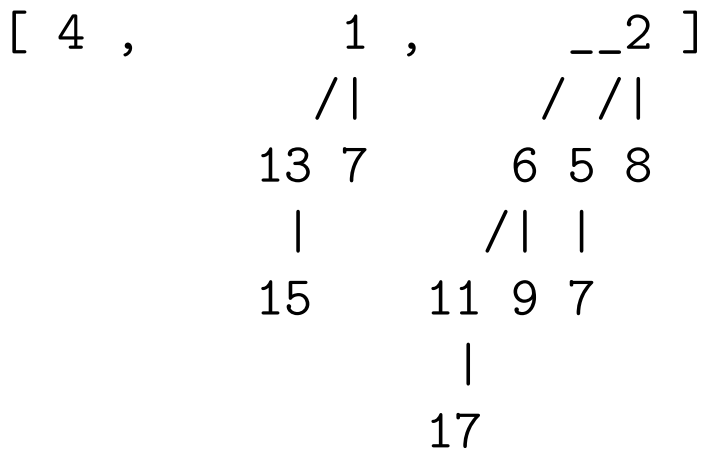
▲ Napište rekurzivní proceduru, která vypíše hodnoty všech uzlů zadaného binárního stromu s n uzly v čase $\mathcal{O}(n)$.

▲ Napište nerekurzivní proceduru, která vypíše hodnoty všech uzlů zadaného binárního stromu s n uzly v čase $\mathcal{O}(n)$ s využitím zásobníku.

▲ Napište proceduru, která ověří, zda daný binární strom je také binární haldou. Předpokládejte, že máme implementovanou funkci `get_length(node, min_length, max_length)`, která pro zadaný uzel `node` vratí délku minimální (`min_length`) a maximální (`max_length`) větve ze zadaného uzlu.

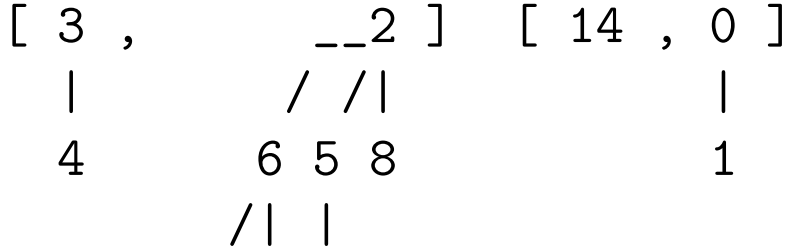
▲ Graficky znázorněte binomiální haldu o 13-ti prvcích.

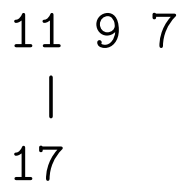
▲ Vložte prvek 3 do binomiální haldy



Jaká je časová složitosť vložení prvku do hlady?

▲ Sloučte binomiální hlady:

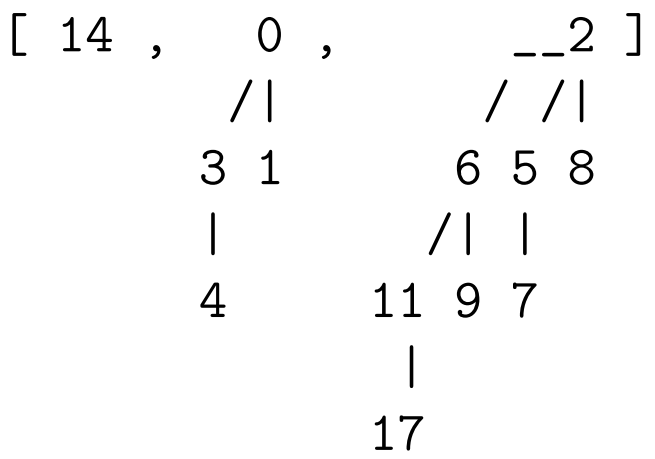




Jaká je časová zložitosť sloučení dvou hlad?

▲ Jak lze nalézt minimální klíč v binomiální haldě? S jakou časovou složitostí?

▲ Odeberte minimum z binomiální hlady



Jaká je časová složitost tohoto algoritmu?