04. Halda, fronta a zásobník

Obsah

Zásobník (Stack)

Zásobník je lineární datová struktura, která pracuje na principu LIFO (Last In, First Out) - poslední dovnitř, první ven.

Vlastnosti zásobníku

- Přidává a odebírá prvky pouze z jednoho konce (vrchol zásobníku)
- Přístup je možný jen k vrchnímu prvku
- Neexistuje přímý přístup k ostatním prvkům
- Neomezený počet prvků (jen paměť)

Reprezentace v programu

Zásobník lze implementovat pomocí: 1. **Pole** - jednoduché, ale s pevnou velikostí 2. **Spojového seznamu** - dynamická velikost, ale větší režie

Základní operace

- **Push** - přidání prvku na vrchol zásobníku
- **Pop** - odebrání prvku z vrcholu zásobníku
- **Peek/Top** - náhled na vrchní prvek bez jeho odebrání
- **isEmpty** - kontrola, zda je zásobník prázdný

Časová složitost

Všechny základní operace mají časovou složitost O(1) - konstantní.

Vhodné užití

- Zpracování syntaxe v kompilátorech (např. kontrola závorek)
- Prohledávání do hloubky (DFS)
- Backtracking algoritmy
- Vyhodnocení výrazů (postfixová/prefixová notace)
- Historie operací (undo/redo)
- Volání funkcí (zásobník volání)

Úskalí

- Přístup pouze k vrchnímu prvku
- Nemožnost iterovat bez destrukce
- Stack overflow - přetečení při příliš mnoha prvcích

Fronta (Queue)

Fronta je lineární datová struktura, která pracuje na principu FIFO (First In, First Out) - první dovnitř, první ven.

Vlastnosti fronty

- Přidává prvky na jeden konec (zadní) a odebírá z druhého konce (přední)
- Přístup jen k přednímu prvku
- Spravedlivé zpracování (kdo dřív přijde, je dřív na řadě)

Reprezentace v programu

Frontu lze implementovat pomocí: 1. **Pole** (kruhové pole pro efektivitu) 2. **Spojového seznamu** (obousměrný pro efektivní přidávání i odebírání)

Základní operace

- **Enqueue** - přidání prvku na konec fronty
- **Dequeue** - odebrání prvku ze začátku fronty
- **Peek/Front** - náhled na první prvek bez jeho odebrání
- **isEmpty** - kontrola, zda je fronta prázdná

Časová složitost

Všechny základní operace mají časovou složitost O(1).

Druhy front

- **Standardní fronta** - FIFO princip
- **Prioritní fronta** - prvky jsou řazeny podle priority (implementováno haldou)
- **Deque** (double-ended queue) - umožňuje přidávat a odebírat z obou konců

Vhodné užití

- Zpracování požadavků (tiskárna, CPU)
- Prohledávání do šířky (BFS)
- Vyrovnávací paměť (buffer)
- Plánování procesů v operačních systémech
- Implementace ostatních datových struktur

Úskalí

- Při implementaci pomocí pole potřeba řešit kruhovou frontu
- Nemožnost náhodného přístupu

Halda (Heap)

Halda je speciální stromová struktura, která splňuje tzv. haldovou vlastnost.

Vlastnosti haldy

- **Min-heap**: každý uzel má hodnotu menší nebo rovnou svým potomkům
- **Max-heap**: každý uzel má hodnotu větší nebo rovnou svým potomkům
- Kompletní binární strom - všechny úrovně jsou plně zaplněny, kromě poslední, která je zaplněna zleva doprava
- Efektivní nalezení minima/maxima (vždy kořen)

Reprezentace v programu

Haldu lze implementovat pomocí: 1. **Pole** - efektivní implicitní reprezentace 2. **Stromové struktury** - méně běžné, větší režie

Při reprezentaci polem: - Kořen je na indexu 0 (nebo 1, podle implementace) - Pro prvek na indexu i: - Levý potomek: 2*i* + *1* - *Pravý potomek: 2*i + 2 - Rodič: (i-1)/2

Základní operace

- **Insert** - vložení nového prvku
- **ExtractMin/ExtractMax** - odebrání a vrácení kořene (min/max hodnoty)
- **Peek** - náhled na kořen bez jeho odebrání
- **Heapify** - přeuspořádání haldy po změně

Vložení prvku do haldy

1. Přidáme prvek na konec haldy
2. "Probubláváme" prvek nahoru, dokud není splněna haldová vlastnost

Odebrání kořene

1. Vyjmeme kořen
2. Nahradíme kořen posledním prvkem haldy
3. "Probubláváme" nový kořen dolů, dokud není splněna haldová vlastnost

Časová složitost

- **Insert**: O(log n)
- **ExtractMin/ExtractMax**: O(log n)
- **Peek**: O(1)
- **Build Heap** (vytvoření haldy z pole): O(n)

Vhodné užití

- Prioritní fronty
- Řadicí algoritmus Heapsort
- Nalezení k největších/nejmenších prvků
- Algoritmy pro hledání cest (např. Dijkstrův algoritmus)
- Huffmanovo kódování

Úskalí

- Nemožnost efektivně vyhledávat jiné než maximální/minimální prvky
- Není vhodná pro sekvenční přístup ke všem prvkům

Porovnání datových struktur

Operace	Zásobník	Fronta	Halda
Vložení	O(1)	O(1)	O(log n)
Odebrání	O(1)	O(1)	O(log n)
Přístup k min/max	O(n)	O(n)	O(1)
Iterace	Nutné vyprázdnit	Nutné vyprázdnit	Netriviální

Implementace v praxi

Zásobník v pseudokódu

```
class Stack:
    data = []  // pole nebo spojový seznam

    push(item):
        data.add(item)

    pop():
```

```
        if isEmpty():
            throw EmptyStackException
        return data.removeLast()

    peek():
        if isEmpty():
            throw EmptyStackException
        return data.getLast()

    isEmpty():
        return data.size() == 0
```

Fronta v pseudokódu

```
class Queue:
    data = []  // pole nebo spojový seznam

    enqueue(item):
        data.addLast(item)

    dequeue():
        if isEmpty():
            throw EmptyQueueException
        return data.removeFirst()

    peek():
        if isEmpty():
            throw EmptyQueueException
        return data.getFirst()

    isEmpty():
        return data.size() == 0
```

Halda v pseudokódu (min-heap)

```
class MinHeap:
    data = []  // pole

    insert(item):
        data.add(item)
        bubbleUp(data.size() - 1)

    extractMin():
        if isEmpty():
            throw EmptyHeapException
        min = data[0]
        data[0] = data.removeLast()
        if !isEmpty():
            bubbleDown(0)
        return min

    bubbleUp(index):
        parent = (index - 1) / 2
        if index > 0 and data[parent] > data[index]:
            swap(parent, index)
            bubbleUp(parent)

    bubbleDown(index):
```

```
        smallest = index
        left = 2 * index + 1
        right = 2 * index + 2

        if left < data.size() and data[left] < data[smallest]:
            smallest = left

        if right < data.size() and data[right] < data[smallest]:
            smallest = right

        if smallest != index:
            swap(index, smallest)
            bubbleDown(smallest)

    isEmpty():
        return data.size() == 0
```

Příklady použití v reálných aplikacích

Zásobník

- **Undo/Redo** v textových editorech
- **Navigace zpět/vpřed** v prohlížečích
- **Vyhodnocení aritmetických výrazů**
- **Systém volání funkcí** v programech

Fronta

- **Tiskové úlohy** v tiskárnách
- **Zpracování požadavků** v serverech
- **Simulace front** v reálném světě
- **Zpracování událostí** v GUI aplikacích

Halda

- **Plánování procesů** v operačních systémech
- **Huffmanovo kódování** pro kompresi dat
- **Algoritmy hledání nejkratší cesty** (Dijkstra)
- **Top K problém** - nalezení K největších/nejmenších prvků

Shrnutí

- **Zásobník** - LIFO, vhodný pro zpětné procházení a rekurzivní algoritmy
- **Fronta** - FIFO, vhodná pro sekvenční zpracování a BFS algoritmy
- **Halda** - efektivní pro prioritní přístup a řazení, vhodná pro algoritmy vyžadující rychlý přístup k minimálním/maximálním hodnotám