PARADIGMA, No.4, 2025, səh.119-124
https://doi.org/10.30546/30015.2025.01.4.1052

STEK (STACK) VERİLƏNLƏR STRUKTURUNUN ARXİTEKTURASI, ƏMƏLİYYATLARI VƏ TƏTBİQ SAHƏLƏRİNİN TƏHLİLİ

C.K. Kazımov, L.Ş. Şirinova
Azərbaycan Universiteti, Ceyhun Hacıbəyli, 71, Bakı, Azərbaycan
e-mail: javanshir.kazimov@au.edu.az,
leyla.shirinova@student.au.edu.az

Xülasə. Məqalədə proqramlaşdırma və məlumatların emalı sistemlərində istifadə edilən stek (stack) verilənlər strukturunun arxitekturası, əsas əməliyyatları və tətbiq sahələri təhlil olunur. Stekin LIFO (Last In, First Out) prinsipi ilə işləməsi, məlumatların yalnız bir ucundan əlavə olunması və eləcədə çıxarılması onun sadə,amma yüksək performanslı struktura çevrilməsinə imkan yaradır. Stekin tətbiq sahələri məqalədə geniş şəkildə araşdırılır: rekursiv funksiyaların icrası, kompilyatorlarda sintaktik analiz, postfix ifadələrin emalı və yaddaşın avtomatik idarə olunması kimi proseslərdə stekin rolu göstərilir. Proqramlaşdırma dillərində stekin massiv və linked-list üsulu ilə reallaşdırılması nümunələri də təhlilə daxil olunmuşdur (C dili ilə). Aparılan tədqiqat göstərir ki, stek sadə strukturu, səmərəli yaddaş istifadəsi və yüksək məhsuldarlığı ilə proqram təminatında mühüm əhəmiyyətə malikdir.

Açar sözlər: Stack, Data structure, LIFO, Push, Pop.

Giriş

İnformasiya texnologiyalarının sürətli inkişafı həm verilənlərin daha effektiv şəkildə saxlanmasını zəruri edir, həm də idarə olunması və emal edilməsini zəruri edir. Proqram təminatının işini yüksəltmək üçün məlumatların daha səmərəli strukturlarda yerləşdirilməsi vacib rol oynayır. Burada verilənlərin strukturları məlumatların məntiqi, ardıcıl və optimallaşdırılmış vəziyyətdə saxlanmasını təmin etməklə yaddaşdan səmərəli istifadənin sürətini artırır. Bu zaman informasiya axtarışı və emalı prosesləri də sürətlənir. Məlumat strukturlarının əsaslarından biri olan **stek (stack)** — son daxil olan və ilk çıxar (LIFO) prinsipi ilə işləyən xüsusi məlumat quruluşu hesab edilir. Steklərin tətbiq sahələri olduqca genişdir, bunlar aşagidakilardır: Kompilyatorlar, proqram icrası zamanı funksiya çağırışları, məntiqi ifadələrin hesablanması, yaddaşın avtomatik idarə edilməsi və daha bir çox alqoritm stek quruluşu üzərində qurula bilir. Stek üzərində aparılan əməliyyatların düzgün alqoritmik reallaşdırılması proqramlaşdırmada mühüm yer tutur.(push, pop, peek, isEmpty, isFull) Bu əməliyyatların hər biri vaxt mürəkkəbliyinə malik olduğundan stek yüksək produktiv sistemlərdə geniş istifadə edilir.

Tədqiqat

Məqalədə stek verilənlər strukturunun nəzəri əsasları, üzərində aparılan əməliyyatların alqoritmləri, tətbiq sahələri və alqoritmik təhlili ətraflı şəkildə araşdırılır. Stek — verilənlərin

yalnız bir ucundan (üst/top hissəsindən) əlavə edildiyi həmçinin də çıxarıldığı ardıcıl olan məlumat strukturudur [1]**.** Stekin prinsipi **LIFO** (Last In, First Out) modelidir.

Demək ki, stekə son daxil olan element ilk öncə çıxarılır. Məsələn, boşqabların bir-birinin üzərinə yığılması kimi: ən sonda qoyulan boşqab masaya götürüləndə birinci götürülür.

Stekin əsas strukturları:

Stek iki əsas üsulla reallaşdırılır [2, 3]**:**

1) Massiv əsasında stek (Array-based stack)

Sabit ölçülü.

Yaddaş ardıcıl olaraq ayrılır.

push və pop əməliyyatları çox sürətli olur.

2) Linked List əsasında stek

- Ölçü dinamik olur.
- Yaddaş lazım olduqca ayrılır.
- Ötürücülər (pointer) vasitəsilə işləyir.

Stekin əsas elementləri

1. **top** — stekin yuxarı hissəsini göstərən göstərici.
2. **stack** — verilənlərin saxlandığı massiv və ya düyün siyahısı.
3. **maxSize** — massiv əsasında steklərdə ölçü limiti.

2.Stek əməliyyatlari və alqoritmləri

Stek üzərində beş əsas əməliyyat yerinə yetirilir [1, 3, 5]:

1. **isEmpty** – stekin boş olduğunu müəyyən etmə
2. **isFull** – stekin dolu olduğunu müəyyən etmə (yalnız massiv üsulunda)
3. **push** – element əlavə etmə
4. **peek/top** – üstdəki elementi yoxlama
5. **pop** – element çıxarma

Push əməliyyatı-vaxt mürəkkəbliyi O(1)

Yeni element stekin yuxarısında yerləşdirilir.

Alqoritm:

1. Əgər stek doludursa → səhv mesajı qaytar.
2. top dəyişənini 1 artır.
3. Yeni elementi stack[top] mövqeyinə yaz.

Pop əməliyyatı-vaxt mürəkkəbliyi O(1)

Stekdəki ən yuxarı elementi çıxarır və geri qaytarır.

Alqoritm:

1. Əgər stek boşdursa → səhv mesajı qaytar.
2. top mövqeyindəki elementi yadda saxla.
3. top dəyişənini 1 azalt.
4. Elementi qaytar.

Peek əməliyyatı-vaxt mürəkkəbliyi O(1)

Stekin yuxarı hissəsindəki elementi oxuyur, lakin çıxarmır.

isEmpty və isFull

isEmpty:

if (top == -1) stek boşdur

isFull:

if (top == maxSize - 1) stek doludur

Stekin tətbiq sahələri

Stek proqram mühəndisliyində ən geniş tətbiq olunan strukturlardan biri hesab olunur.

Rekursiv funksiyaların icrası [1, 6]:

Rekursiya zamanı hər funksiya çağırışı sistem stekinə push edilir. Funksiya bitdikdə stekdən pop edilir.

Kompilyatorlarda istifadə **[6]:**

- postfix ifadələrin hesablanması
- sintaktik analiz
- operatorların öncüllüyü
- mötərizələrin düzgünlüyünün yoxlanması

Yaddaşın avtomatik idarə olunması

Proqramların lokal dəyişənləri stek yaddaşında saxlanır. Funksiya bitdikdən sonra həmin yaddaş avtomatik azad edilir.

Backtracking (geri qayıtma) alqoritmləri

Stek istifadə olunur:

- labirint tapmacalarında
- DFS qraf axtarışında

3.Stekin üzərində aparılan əməliyyatların proqram təminati

Aşağıda steklərin C dilində reallaşdırılması nümunələri təqdim olunur .

Massiv əsasında stek (C kodu) olan halda:

```
#include <stdio.h>
#define MAX 100
int stack[MAX];
```

```
int top = -1;
void push(int x) {
if (top == MAX - 1) {
printf("Stek doludur!\n");
} else {
top++;
stack[top] = x;
}
}
int pop() {
if (top == -1) {
printf("Stek boşdur!\n");
return -1;
} else {
return stack[top--];
}
}
int peek() {
if (top == -1) {
printf("Stek boşdur!\n");
return -1;
}
return stack[top];
}
int main() {
push(10);
push(20);
push(30);
printf("Üstdəki element: %d\n", peek());
printf("Çıxarılan element: %d\n", pop());
return 0;
}
```

Linked List əsasında stek olan halda:

```
#include <stdio.h>
```

```
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* top = NULL;
void push(int x) {
struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
temp->data = x;
temp->next = top;
top = temp;
}
int pop() {
if (top == NULL) {
printf("Stek boşdur!\n");
return -1;
}
int val = top->data;
struct Node* temp = top;
top = top->next;
free(temp);
return val;
}
```

Stekin üstünlükləri və çatişmazliqlari

Üstünlüklər

- Əməliyyatlar O(1) vaxt mürəkkəbliyinə malikdir.
- Sadə quruluşlu və tətbiqi asandır.
- Rekursiyanın reallaşdırılmasının əsasıdır.
- Yaddaşdan səmərəli istifadə imkanı yaradır.

Çatışmazlıqlar

- Sadəcə bir ucundan əməliyyat aparmaq olur.
- Massiv əsasında steklərin ölçüsü sabitdir.
- Təsadüfi giriş mümkün deyil.

Nəticə

Stek verilənlər strukturu kompüter elmləri və proqram təminatı mühəndisliyinin ən fundamental elementlərindən biridir. Onun rekursiya, kompilyasiya, yaddaşın idarə olunması, qraf axtarış alqoritmləri, riyazi ifadələrin emalı kimi sahələrdə geniş tətbiqi təhlil edilir və əməliyyatların alqoritm və proqram təminatı verilir.Stek əməliyyatlarının sadəliyi və yüksək performansı bu məlumat quruluşunu proqram sistemlərinin ayrılmaz hissəsinə çevirir.

Ədəbiyyat

1. Cormen T., Leiserson C., Rivest R., Stein C., (2009), Introduction to algorithms, Cambridge, MIT Press.
2. Weiss M. A., (2013), Data structures and algorithm analysis in *C*, Boston, Pearson Education.
3. Goodrich M. T., Tamassia R., (2011), Data structures and algorithms in C++, Hoboken, Wiley.
4. Knuth D. E., (1997), The art of computer programming. Vol. 1: Fundamental algorithms, Reading, Addison-Wesley.
5. Lafore R., (2002), Data structures and algorithms in Java, Indianapolis, Sams Publishing.
6. Sedgewick R., Wayne K., (2011), Algorithms, Boston, Addison-Wesley.

ANALYSIS OF THE ARCHITECTURE, OPERATIONS AND APPLICATION AREAS OF THE STACK DATA STRUCTURE

C.K. Kazimov, L.Sh. Shirinova
Azerbaijan University, Ceyhun Hajibeyli, 71, Baku, Azerbaijan

Abstract: This article analyzes the architecture, main operations, and application areas of the stack data structure used in programming and information processing systems. The stack operates on the LIFO (Last In, First Out) principle, allowing data to be added and removed from only one end, making it a simple yet high-performance structure. The application areas of the stack are extensively explored: execution of recursive functions, syntax analysis in compilers, evaluation of postfix expressions, and automatic memory management. Examples of stack implementation using arrays and linked lists in programming languages, including C, are also included in the analysis. The study demonstrates that the stack, with its simple structure, efficient memory usage, and high performance, plays a critical role in software development.

Keywords: Stack, Data structure, LIFO, Push, Pop.