Comparatie stiva si coada

**Stiva**

Prima implementare e stiva prin lista simpla inlantuita:

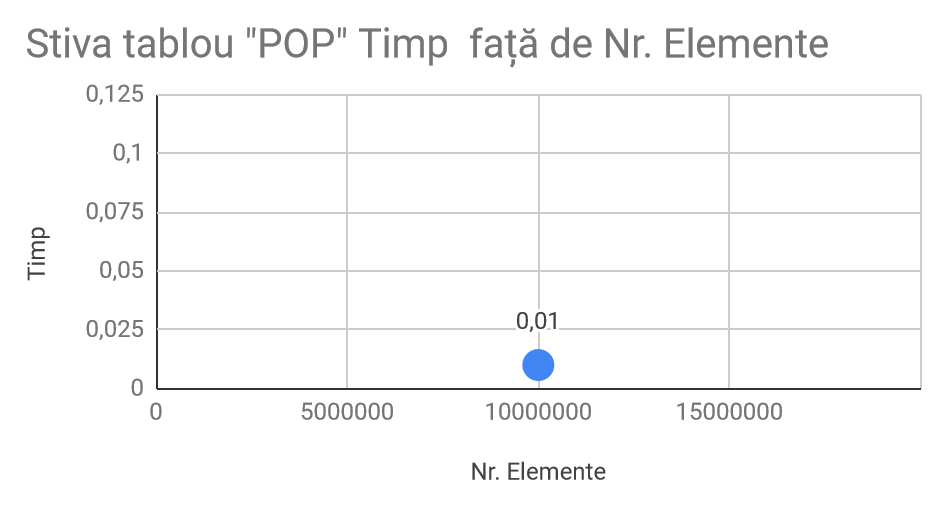
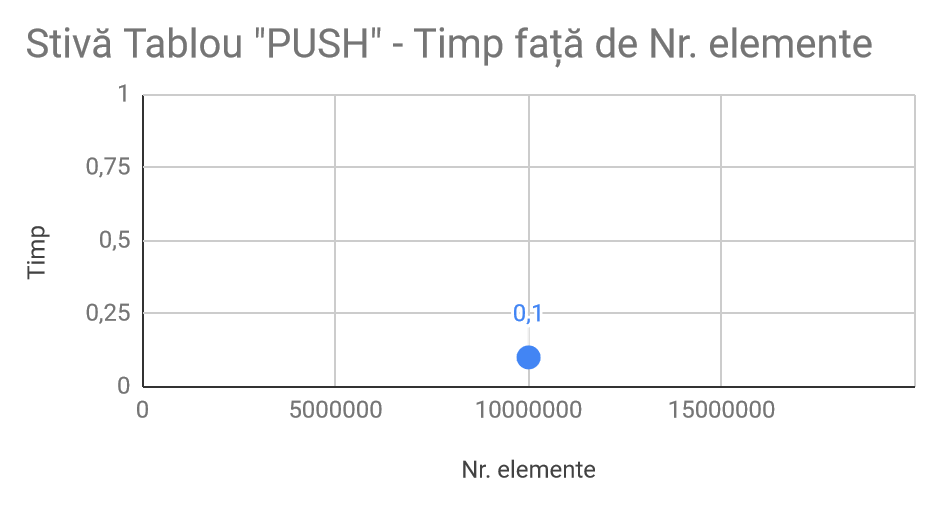
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <limits.h>
4. #include <time.h>
6. struct StackNode
7. {
8. int data;
9. struct StackNode\* next;
10. };
12. struct StackNode\* newNode(int data)
13. {
14. struct StackNode\* stackNode =
15. (struct StackNode\*) malloc(sizeof(struct StackNode));
16. stackNode->data = data;
17. stackNode->next = NULL;
18. return stackNode;
19. }
21. int isEmpty(struct StackNode \*root)
22. {
23. return !root;
24. }
26. void push(struct StackNode\*\* root, int data)
27. {
28. struct StackNode\* stackNode = newNode(data);
29. stackNode->next = \*root;
30. \*root = stackNode;
31. printf("%d pushed to stack\n", data);
32. }
34. int pop(struct StackNode\*\* root)
35. {
36. if (isEmpty(\*root))
37. return INT\_MIN;
38. struct StackNode\* temp = \*root;
39. \*root = (\*root)->next;
40. int popped = temp->data;
41. free(temp);
43. return popped;
44. }
46. int peek(struct StackNode\* root)
47. {
48. if (isEmpty(root))
49. return INT\_MIN;
50. return root->data;
51. }
53. int main()
54. {
56. struct StackNode\* root = NULL;
57. int n=1000;
58. clock\_t start,end;
59. double total;
60. int i;
62. for(i=0;i<n;i++)
64. {push(&root,i);}
65. start=clock();
66. for(i=0;i<n;i++)
67. {
68. pop(i);
69. }
70. end=clock();
71. total=(double)(end-start)/CLOCKS\_PER\_SEC;

74. printf("%d popped from stack\n", pop(&root));
76. printf("Top element is %d\n", peek(root));
77. printf("%.2f",total);
79. return 0;
80. }

**Timpi de executie:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Stiva tablou | Operatiuni | Timp | Nr. elemente |
|  | Push | 0,00 | 100 |
|  |  | 0,00 | 1000 |
|  |  | 0,00 | 10000 |
|  |  | 0,00 | 100000 |
|  |  | 0,01 | 10000000 |
|  |  | 0,08 | 10 milioane |
|  | Pop | 0,00 | 100 |
|  |  | 0,00 | 1000 |
|  |  | 0,00 | 10000 |
|  |  | 0,00 | 100000 |
|  |  | 0,01 | 10000000 |
|  |  | 0,12 | 10 milioane |

**Grafic:**



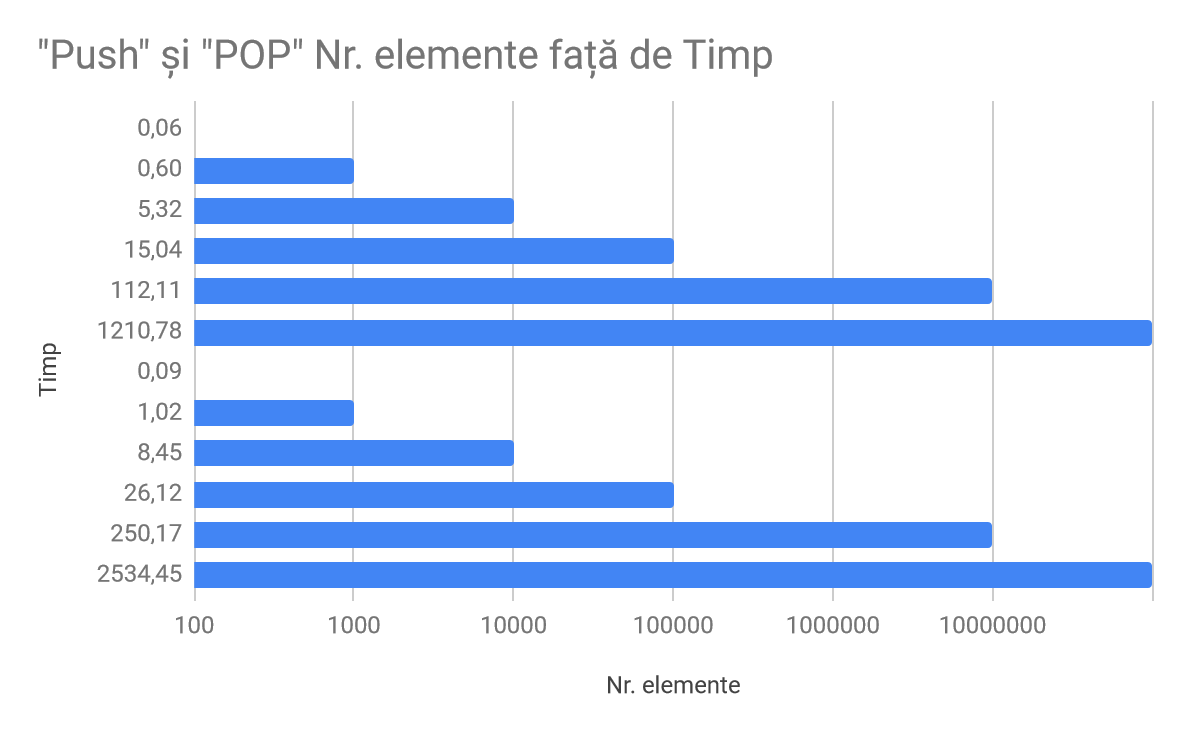
A doua implementare este stiva prin tablou:

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <limits.h>
4. #include <time.h>
5. struct stiva
6. {
7. int varf;
8. unsigned capacitate;
9. int\* m;
10. };
11. struct stiva\* createstiva(unsigned capacitate)
12. {
13. struct stiva\* stiva=(struct stiva\*) malloc(sizeof(struct stiva));
14. stiva -> capacitate = capacitate;
15. stiva-> varf= -1;
16. stiva -> m = (int\*) malloc(stiva->capacitate\* sizeof(int ));
17. return stiva;
18. };
19. int full(struct stiva\* stiva)
20. {
21. return stiva->varf == stiva->capacitate -1;
22. }
23. int goala(struct stiva\* stiva)
24. {
25. return stiva->varf == -1;
26. }
27. void push(struct stiva\* stiva, int obiect)
28. {
29. if(full(stiva))
30. return;
31. stiva->m[++stiva->varf]= obiect;
33. }
34. int pop(struct stiva\* stiva)
35. {
36. if(goala(stiva))
37. return INT\_MIN;
38. return stiva->m[stiva->varf--];
39. }
40. int main()
41. {
42. int N = 10000000;
43. struct stiva\* stiva = createstiva(N);
44. clock\_t start,end;
46. int i;
47. double total;
48. start=clock();
49. for(i=0;i<N;i++)
50. {
51. push(stiva,i);
52. }
54. for(i=0;i<N;i++)
55. pop(stiva);
56. end=clock();
57. total= (double)(end-start)/CLOCKS\_PER\_SEC;
58. printf("%.2f", total);
60. return 0;
61. }

**Timpi de exectie:**

|  |  |  |  |
| --- | --- | --- | --- |
| Stiva lista | Operatiuni | Timp | Nr. elemente |
|  | Push | 0,06 | 100 |
|  |  | 0,60 | 1000 |
|  |  | 5,32 | 10000 |
|  |  | 15,04 | 100000 |
|  |  | 112,11 | 10000000 |
|  |  | 1210,78 | 100000000 |
|  | Pop | 0,09 | 100 |
|  |  | 1,02 | 1000 |
|  |  | 8,45 | 10000 |
|  |  | 26,12 | 100000 |
|  |  | 250,17 | 10000000 |
|  |  | 2534,45 | 100000000 |

**Grafic:**



Dupa cum putem observa din tabel si grafic stiva implementata prin tablou este mult mai rapida fata de stiva implementa prin lista simpla.

**Coada**

Prima implementare este coada prin lista simpla:

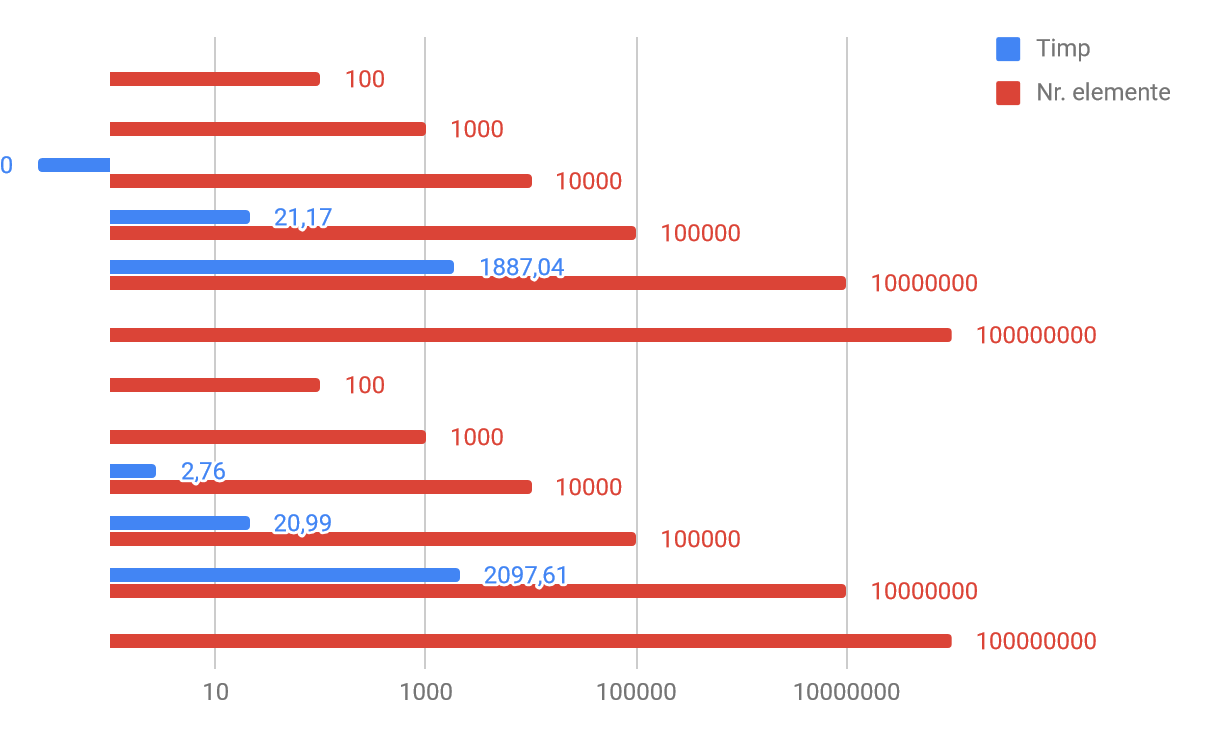
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. typedef struct Node {
5. int value;
6. struct Node \*next;
7. } Node;
8. typedef struct DinamicQueue {
9. int top, max;
10. Node \*head;
11. } DinamicQueue;
13. Node \*create\_node(int value)
14. {
15. Node \*node = malloc(sizeof(Node));
16. node->next = NULL;
17. node->value = value;
19. return node;
20. }
22. DinamicQueue \*create\_dinamic\_queue(int max)
23. {
24. DinamicQueue \*dinamicQueue = malloc(sizeof(DinamicQueue));
26. dinamicQueue->top = 0;
27. dinamicQueue->max = max;
28. dinamicQueue->head = NULL;
30. return dinamicQueue;
31. }
33. void push(DinamicQueue \*dinamicQueue, int value)
34. {
35. if (dinamicQueue->head == NULL) {
36. dinamicQueue->head = create\_node(value);
37. } else {
38. Node \*tmp = dinamicQueue->head;
39. while (tmp->next != NULL)
40. {
41. tmp = tmp->next;
42. }
43. tmp->next = create\_node(value);
44. }
45. }
46. int pop\_dinamic\_queue(DinamicQueue \*dinamicQueue)
47. {
48. int value = dinamicQueue->head->value;
49. Node \*temp = dinamicQueue->head;
50. dinamicQueue->head = dinamicQueue->head->next;
51. free(temp);
53. return value;
54. }
55. int main()
56. {
57. clock\_t start,end;
58. int n=1000000;
59. int i;
60. double total;
61. DinamicQueue \*dinamicQueue = create\_dinamic\_queue(n);
62. for(i=0;i<n;i++)
63. {
64. push(dinamicQueue,i);
65. }
67. start=clock();
68. for(i=0;i<n;i++)
69. {
70. pop\_dinamic\_queue(dinamicQueue);
71. }
72. end=clock();
73. total=(double)(end-start)/CLOCKS\_PER\_SEC;
74. printf("%.2f",total);

77. //printf("%d ", pop\_dinamic\_queue(dinamicQueue));
78. //printf("%d ", pop\_dinamic\_queue(dinamicQueue));
79. //printf("%d ", pop\_dinamic\_queue(dinamicQueue));
81. return 0;
82. }

**Timpi de executie:**

|  |  |  |  |
| --- | --- | --- | --- |
| Coada tablou |  | Timp | Nr. elemente |
|  | Push | 0,00 | 100 |
|  |  | 0,00 | 1000 |
|  |  | 0,20 | 10000 |
|  |  | 21,17 | 100000 |
|  |  | 1887,04 | 10000000 |
|  |  | Very long | 100000000 |
|  | Pop | 0,00 | 100 |
|  |  | 0,00 | 1000 |
|  |  | 2,76 | 10000 |
|  |  | 20,99 | 100000 |
|  |  | 2097,61 | 10000000 |
|  |  | Very long | 100000000 |

**Grafic:**



A doua implementare e coada prin tablou:

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <limits.h>
4. #include <time.h>
6. struct Queue
7. {
8. int front, rear, size;
9. unsigned capacity;
10. int\* array;
11. };


15. struct Queue\* createQueue(unsigned capacity)
16. {
17. struct Queue\* queue = (struct Queue\*) malloc(sizeof(struct Queue));
18. queue->capacity = capacity;
19. queue->front = queue->size = 0;
20. queue->rear = capacity - 1;
21. queue->array = (int\*) malloc(queue->capacity \* sizeof(int));
22. return queue;
23. }

26. int isFull(struct Queue\* queue)
27. { return (queue->size == queue->capacity); }

30. int isEmpty(struct Queue\* queue)
31. { return (queue->size == 0); }

34. void enqueue(struct Queue\* queue, int item)
35. {
36. if (isFull(queue))
37. return;
38. queue->rear = (queue->rear + 1)%queue->capacity;
39. queue->array[queue->rear] = item;
40. queue->size = queue->size + 1;
41. printf("%d enqueued to queue\n", item);
42. }

45. int dequeue(struct Queue\* queue)
46. {
47. if (isEmpty(queue))
48. return INT\_MIN;
49. int item = queue->array[queue->front];
50. queue->front = (queue->front + 1)%queue->capacity;
51. queue->size = queue->size - 1;
52. return item;
53. }

56. int front(struct Queue\* queue)
57. {
58. if (isEmpty(queue))
59. return INT\_MIN;
60. return queue->array[queue->front];
61. }

64. int rear(struct Queue\* queue)
65. {
66. if (isEmpty(queue))
67. return INT\_MIN;
68. return queue->array[queue->rear];
69. }

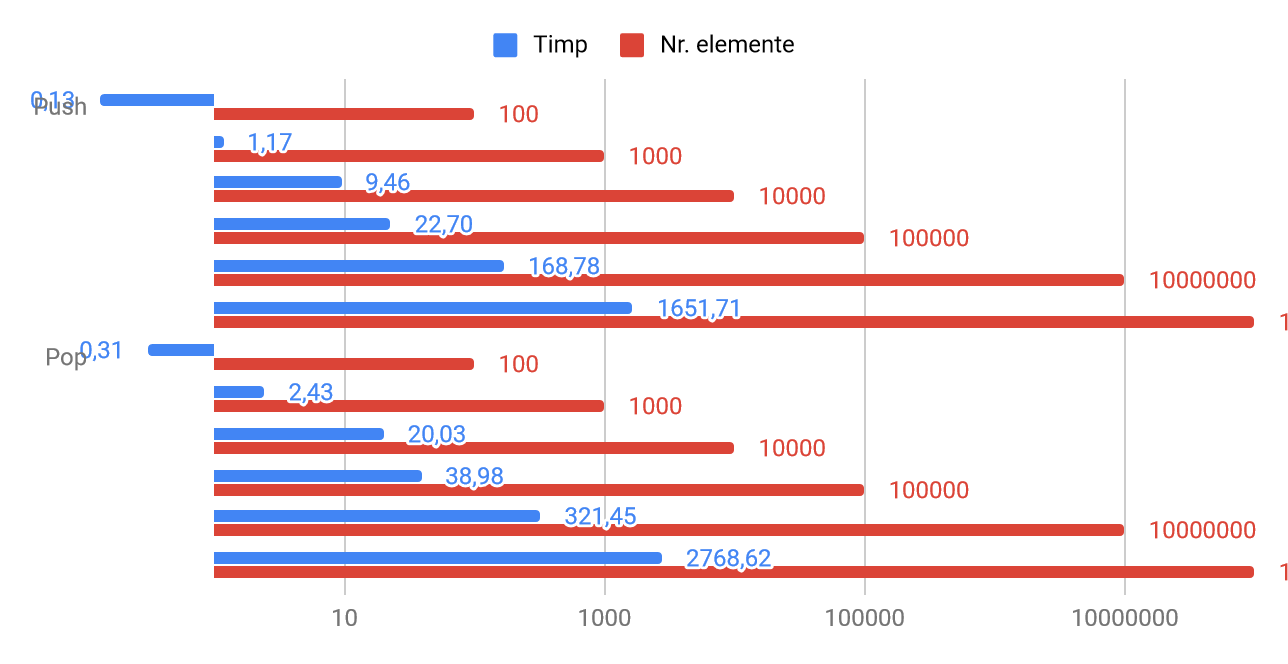
72. int main()
73. {
74. int n=10000;
75. struct Queue\* queue = createQueue(n);
76. clock\_t start,end;
77. int i;
78. double total;
79. start=clock();
80. for(i=0;i<n;i++)
81. {
82. enqueue(queue,i);
83. }
84. end=clock();

87. total=(double)(end-start)/CLOCKS\_PER\_SEC;
88. printf("%d dequeued from queue\n\n", dequeue(queue));
90. printf("Front item is %d\n", front(queue));
91. printf("Rear item is %d\n", rear(queue));
92. printf("%.2f",total);
94. return 0;
95. }

**Timpi de executie:**

|  |  |  |  |
| --- | --- | --- | --- |
| Coada tablou |  | Timp | Nr. elemente |
|  | Push | 0,13 | 100 |
|  |  | 1,17 | 1000 |
|  |  | 9,46 | 10000 |
|  |  | 22,70 | 100000 |
|  |  | 168,78 | 10000000 |
|  |  | 1651,71 | 100000000 |
|  | Pop | 0,31 | 100 |
|  |  | 2,43 | 1000 |
|  |  | 20,03 | 10000 |
|  |  | 38,98 | 100000 |
|  |  | 321,45 | 10000000 |
|  |  | 2768,62 | 100000000 |

**Grafic:**



La coada in schimb este invers coada implementata prin lista simpla este mai rapida in introducerea si extragerea de elemente fata de coada implementata prin tablou.

**Bibliografie:**

<https://www.geeksforgeeks.org/>

i