Innbyggðar gagnagrindur

Bergur Snorrason

January 15, 2024

Innbyggðar gagnagrindur í `C++`

- Grunnur `C++` býr yfir mörgum öflugum gagnagrindum.

Innbyggðar gagnagrindur í `C++`

- Grunnur `C++` býr yfir mörgum öflugum gagnagrindum.
- Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.

Innbyggðar gagnagrindur í `C++`

- Grunnur `C++` býr yfir mörgum öflugum gagnagrindum.
- Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.
- Við munum bara fjalla um gagnagrindurnar í grófum dráttum.

Innbyggðar gagnagrindur í `C++`

- Grunnur `C++` býr yfir mörgum öflugum gagnagrindum.
- Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.
- Við munum bara fjalla um gagnagrindurnar í grófum dráttum.
- Það er hægt að finna ítarlegra efni og dæmi um notkun á netinu.

Fylki

- Lýkt og í mörgum öðrum forritunarmálum eru fylki í `C++`.

Fylki

- Lýkt og í mörgum öðrum forritunarmálum eru fylki í `C++`.
- Fylki geyma gögn og eru af fastri stærð.

Fylki

- Lýkt og í mörgum öðrum forritunarmálum eru fylki í `C++`.
- Fylki geyma gögn og eru af fastri stærð.
- Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.

Fylki

- Lýkt og í mörgum öðrum forritunarmálum eru fylki í `C++`.
- Fylki geyma gögn og eru af fastri stærð.
- Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.
- Þetta leyfir manni að vísa í fylkið í $\mathcal{O}(1)$.

Fylki

- Lýkt og í mörgum öðrum forritunarmálum eru fylki í `C++`.
- Fylki geyma gögn og eru af fastri stærð.
- Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.
- Þetta leyfir manni að vísa í fylkið í $\mathcal{O}(1)$.

Aðgerð	Tímaflækja
Lesa eða skrifa ótiltekið stak	$\mathcal{O}(1)$
Bæta staki aftast	$\mathcal{O}(n)$
Skeyta saman tveimur	$\mathcal{O}(n)$

`vector`

- Gagnagrindin `vector` er að mestu leiti eins og fylki.

`vector`

- Gagnagrindin `vector` er að mestu leiti eins og fylki.
- Það má þó bæta stökum aftan á `vector` í $\mathcal{O}(1)$.

`vector`

- Gagnagrindin `vector` er að mestu leiti eins og fylki.
- Það má þó bæta stökum aftan á `vector` í $\mathcal{O}(1)$.
- Margir nota bara `vector` og aldrei fylki sem slík.

`vector`

- Gagnagrindin `vector` er að mestu leiti eins og fylki.
- Það má þó bæta stökum aftan á `vector` í $\mathcal{O}(1)$.
- Margir nota bara `vector` og aldrei fylki sem slík.

Aðgerð	Tímaflækja
Lesa eða skrifa ótiltekið stak	$\mathcal{O}(1)$
Bæta staki aftast	$\mathcal{O}(1)$
Skeyta saman tveimur	$\mathcal{O}(n)$

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    srand(time(NULL));
    int i, n = 10;
    vector<int> a;
    for (i = 0; i < n; i++) a.push_back(rand()%100);
    for (i = 0; i < n; i++) printf("%2d ", a[i]);
    printf("\n");
    sort(a.begin(), a.end());
    for (i = 0; i < n; i++) printf("%2d ", a[i]);
    printf("\n");
    sort(a.rbegin(), a.rend());
    for (i = 0; i < n; i++) printf("%2d ", a[i]);
    printf("\n");
    return 0;
}
```

```
>> ./a.out
83 23 26 24 52 74  0 39  0 90
 0  0 23 24 26 39 52 74 83 90
90 83 74 52 39 26 24 23  0  0
```

`list`

- Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.

`list`

- Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- Því er uppfletting ekki hröð.

`list`

- Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- Því er uppfletting ekki hröð.
- Aftur á móti er hægt að gera smávægilegar breytingar á `list` sem er ekki hægt að gera á fylkjum.

`list`

- Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- Því er uppfletting ekki hröð.
- Aftur á móti er hægt að gera smávægilegar breytingar á `list` sem er ekki hægt að gera á fylkjum.

Aðgerð	Tímaflækja
Finna stak	$\mathcal{O}(n)$
Bæta staki aftast	$\mathcal{O}(1)$
Bæta staki fremst	$\mathcal{O}(1)$
Bæta staki fyrir aftan tiltekið stak	$\mathcal{O}(1)$
Bæta staki fyrir framan tiltekið stak	$\mathcal{O}(1)$
Skeyta saman tveimur	$\mathcal{O}(1)$

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int i;
    list<int> a;
    for (i = 1; i <= 5; i++) a.push_back(i);
    for (i = 1; i <= 5; i++) a.push_front(i);
    for (auto p : a) printf("%d ", p);
    printf("\n");
    return 0;
}
```

```
>> ./a.out
5 4 3 2 1 1 2 3 4 5
```

`stack`

- Gagnagrindin `stack` geymir gögn og leyfir aðgang að síðasta staki sem var bætt við.

`stack`

- Gagnagrindin `stack` geymir gögn og leyfir aðgang að síðasta staki sem var bætt við.

Aðgerð	Tímaflækja
Bæta við staki	$\mathcal{O}(1)$
Lesa nýjasta stakið	$\mathcal{O}(1)$
Fjarlægja nýjasta stakið	$\mathcal{O}(1)$

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int i, n = 10;
    stack<int> a;
    for (i = 0; i < 10; i++) a.push(i);
    while (a.size() > 0)
    {
        printf("%d ", a.top());
        a.pop();
    }
    printf("\n");
    return 0;
}
```

```
>> ./a.out
9 8 7 6 5 4 3 2 1 0
```

`queue`

- Gagnagrindin `queue` geymir gögn og leyfir aðgang að fyrsta stakinu sem var bætt við.

`queue`

- Gagnagrindin `queue` geymir gögn og leyfir aðgang að fyrsta stakinu sem var bætt við.

Aðgerð	Tímaflækja
Bæta við staki	$\mathcal{O}(1)$
Lesa elsta stakið	$\mathcal{O}(1)$
Fjarlægja elsta stakið	$\mathcal{O}(1)$

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int i, n = 10;
    queue<int> a;
    for (i = 0; i < 10; i++) a.push(i);
    while (a.size() > 0)
    {
        printf("%d ", a.front());
        a.pop();
    }
    printf("\n");
    return 0;
}
```

```
>> ./a.out
0 1 2 3 4 5 6 7 8 9
```

`set`

- Gagnagrindin `set` geymir gögn án endurtekninga og leyfir hraða uppflettingu.

`set`

- Gagnagrindin `set` geymir gögn án endurtekninga og leyfir hraða uppflettingu.

Aðgerð	Tímaflækja
Bæta við staki	$\mathcal{O}(\log n)$
Fjarlægja stak	$\mathcal{O}(\log n)$
Gá hvort staki hafi verið bætt við	$\mathcal{O}(\log n)$

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    srand(time(NULL));
    int i, n = 10;
    set<int> a;
    while (a.size() < 10) a.insert(rand()%100);
    while (a.size() > 0)
    {
        printf("%2d ", *a.begin());
        a.erase(a.begin());
    }
    printf("\n");
    return 0;
}
```

```
>> ./a.out
 2  9 18 29 34 42 43 46 91 97
```