Семинар 3 - Вектор. Амортизирана сложност

Вектор

* вид списъчна структура (запазва реда на добавяне на елем.)

→ Основни операции на вектора:

* добавяне в края → push_back() → best-case $\Theta(1)$; worst-case $\Theta(n)$ - когато трябва да се напр. resize
* премахване от края → pop_back() - $\Theta(1)$ /НЕ прави автоматичен resize/
* индексация - $\Theta(1)$
* добавяне на произволна позиция → insert() → $\Theta(n)$
* премахване от произволна позиция → erase() → $\Theta(n)$

→ Защо са важни? - Векторът има <u>locality</u> ⚠

→ Анализ на push_back операцията

* best-case $\Theta(1)$
* worst-case $\Theta(n)$
* average-case ? → НЕ го дефинираме, защото векторът не е случайна величина

Разглеждаме последователност от операции в/у един и същи контекст => амортизиран анализ /интересува ни средно на операция/

При добавяне на n числа:

- бързи стъпки $\Theta(1)$ - когато не правим resize $\Rightarrow \underline{\underline{\leq n}}$
- бавни стъпки $\Theta(n)$ - когато правим resize

на итерации $\underbrace{1, 2, 4, 8, 16 \dots \sim n}_{\approx 2n-1}$ // growth factor = 2 $\underbrace{\sum_{i=0}^{\log_2 n} 2^i}$

=> общо за n добавяния правим $\leq n * \Theta(1) + 2n - 1 \approx \Theta(n)$ стъпки

=> $\Theta(1)$ амортизирана сложност на добавяне

→ Детайли по имплементацията

• Итератор - iterator ; const iterator ; reverse iterator
* генерализация на указател
* служи за обхождане на колекция
* основни операции : op++ , op--, op* , сравнения ==, !=

• Заделяне на памет
1) new expression - заделя памет + вика конструктор
2) operator new - заделя количество памет
3) placement new - вика конструктор в/у заделена вече памет

1) = 2) + 3)

пр. 1) T* data = new T [count]

пр. 2) operator new (count * sizeof(T))
↳ връща void*   ↳ брой байтове

пр. 3) placement new
new (&data[i]) T(...);
адрес в паметта   конструктор

• Освобождаване на памет
* free() - 1) викаме експлицитно деструкторите на създадените обекти (до size)
2) викаме operator delete да освободи паметта
/operator delete(_data, capacity()*sizeof(T));/

↓ по същия начин в reserve()
↳ пълним новия масив с move семантика

• Други особености
* push_back() vs. emplace_back()
↳ приема параметри и създава обект на база тези параметри

```
emplace_back (Args&& ... args) {
    ...
    new (&_data[_size++]) T(std::forward<Args>
                               (args)...);
}
```

* shrink_to_fit()
↳ намаля размера на колекцията, така че да е точно size

Задача. Да се генерират всички булеви вектори с дължина $\underline{\underline{n}}$. Да се направи анализ на функцията (без print!).

```
void generateBoolVectors (int n) {
    std::vector<int> v(n, 0);
    unsigned count = 1 << n;
    for (unsigned i = 0; i < count; i++) {
        print(v);
        next(v);
    }
}

void next (std::vector<int>& v) {
    int idx = v.size() - 1;
    while (idx >= 0 && v[idx] == 1) {
        v[idx--] = 0;
    }
    if (idx >= 0) {
        v[idx] = 1;
    }
}
```

Анализ на next:

* best-case – когато завършва на 0 $\Rightarrow \Theta(1)$
* worst-case – когато са само 1-ци $\Rightarrow \Theta(n)$
* амортизиран анализ

имаме 2^n обекта общо $\Rightarrow$ долна граница $\Omega(2^n)$

разглеждаме while цикъла – условието

false 2^n пъти (когато обектът е почти готов и остава да се изпълни if-а)

true $\sum_{i=0}^{n-1} 2^i$ $\rightarrow$ когато сме на позиция $idx = i$, услов. е вярно точно 2^i пъти

$\underbrace{\boxed{\;\ldots\;}\boxed{1}}_{2^{n-1}\text{ варианта}}$

$\underbrace{\boxed{\;\ldots\;}\boxed{1}\boxed{1}}_{2^{n-2}\text{ варианта}}$ … и т.н.

$\Rightarrow$ общо $2^n + \sum_{i=0}^{n-1} 2^i = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ | на вектор $\approx \frac{2^{n+1}-1}{2^n} \le 2$ стъпки $\Rightarrow \Theta(1)$