

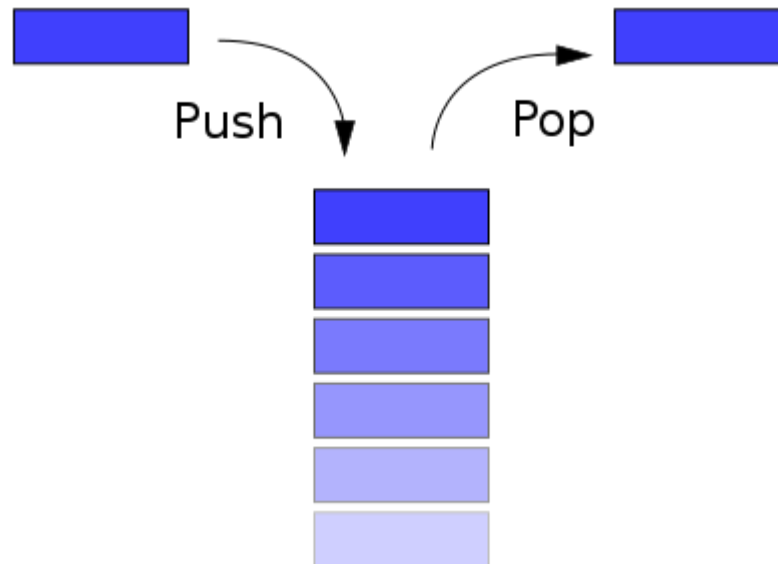
Bufory

Stos

- Stos (ang. stack) to struktura, która pozwala na gromadzenie i przetwarzanie danych zgodnie z kolejnością ich wejścia do struktury: element który został najpóźniej dodany do struktury zostanie z niej najszybciej usunięty (ang. LIFO – Last In, First Out; am. LCFS – Last Come, First Served).
- Analogia: stos książek na biurku, ubrania na krześle .

Stos

- Operacje na stosie:
 - Włożenie elementu na szczyt stosu (ang. **push**)
 - Usunięcie elementu ze szczytu stosu (ang. **pop**)
 - Podglądnięcie elementu na szczycie stosu (ang. **top**)
 - Liczba wszystkich elementów na stosie (ang. **size**)



Implementacja stosu na tablicy

- Do zwykłej tablicy dokładamy dwie informacje: pojemność stosu (liczba slotów w tablicy), zapełnienie stosu (liczba użytych elementów w tablicy):
 - `tab[]` – tablica na dane
 - `poj` – wielkość tablicy
 - `zap` – zapełnienie tablicy

Implementacja stosu na tablicy

- Operacja push(x):
push(x) {
 if (zap = poj) error;
 tab[zap] := x;
 inc(zap);
}
- Operacja pop():
pop() {
 if (zap = 0) error;
 dec(zap);
 return tab[zap];
}

Implementacja stosu na tablicy

- Wszystkie operacje na stosie zaimplementowanym na tablicy zajmują czas stały $O(1)$.

Implementacja stosu na liście jednokierunkowej

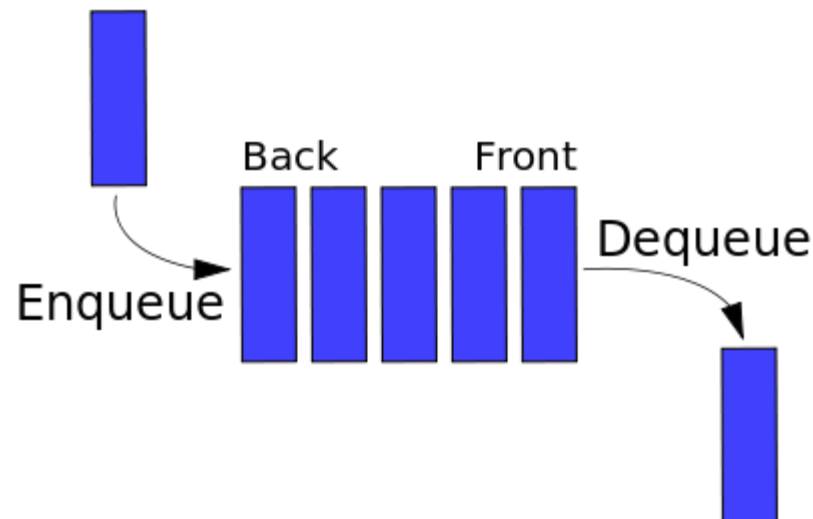
- Operacja wstawiania elementu na stos: dołączenie elementu na początek listy.
- Operacja ściągnięcia elementu ze stosu: usunięcie pierwszego elementu z listy (o ile lista nie jest pusta).
- Wszystkie operacje na stosie zaimplementowanym na liście zajmują czas stały $O(1)$.
- Pojemność listy jest nieograniczona.
- Potrzebujemy trzymać wskaźnik na początek listy.

Kolejka

- Kolejka (ang. queue) to struktura, która pozwala na gromadzenie i przetwarzanie danych zgodnie z kolejnością ich wejścia do struktury: element który został najwcześniej dodany do struktury zostanie z niej najszybciej usunięty (ang. FIFO – first in first out; am. FCFS – First Come, First Served).
- Analogia: kolejka do kasy.

Kolejka

- Operacje na kolejce:
 - Włożenie elementu na koniec kolejki (ang. **enqueue**)
 - Usunięcie elementu z początku kolejki (ang. **dequeue**)
 - Podglądnięcie elementu na początku kolejki (ang. **front**)
 - Liczba wszystkich elementów w kolejce (ang. **size**)



Implementacja kolejki na tablicy

- Do zwykłej tablicy dokładamy trzy informacje: pojemność kolejki (liczba slotów w tablicy), początek kolejki (numer indeksu komórki w którym znajduje się pierwszy element w kolejce), zapełnienie kolejki (liczba użytych elementów w tablicy):
 - `tab[]` – tablica na dane
 - `poj` – wielkość tablicy
 - `zap` – zapełnienie tablicy
 - `pocz` – miejsce, od którego rozpoczyna się kolejka
- Uwaga: zawijamy tablicę.

Implementacja kolejki na tablicy

- Operacja enqueue(x):

```
enqueue(x) {  
    if (zap = poj) error;  
    tab[(pocz + zap) mod poj] := x;  
    inc(zap);  
}
```
- Operacja dequeue():

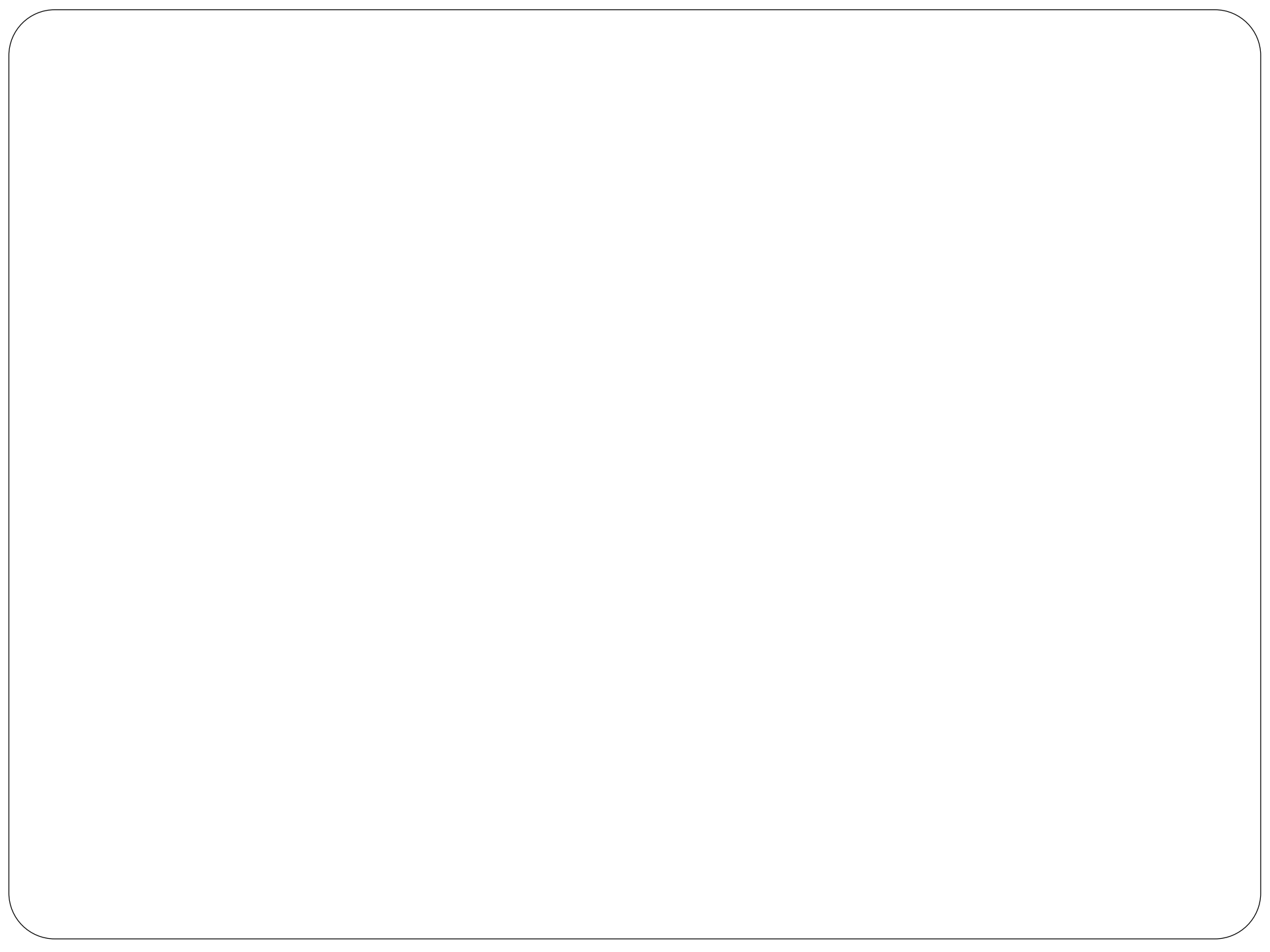
```
dequeue() {  
    if (zap = 0) error;  
    x := tab[(pocz + zap) mod poj];  
    pocz := (pocz + 1) mod poj;  
    dec(zap);  
    return x;  
}
```

Implementacja kolejki na tablicy

- Wszystkie operacje na kolejce zaimplementowanej na tablicy zajmują czas stały $O(1)$.

Implementacja kolejki na liście jednokierunkowej

- Operacja wstawiania elementu do kolejki: dołączenie elementu na koniec listy.
- Operacja ściągnięcia elementu z kolejki: usunięcie pierwszego elementu z listy (o ile lista nie jest pusta).
- Wszystkie operacje na kolejce zaimplementowanej na liście zajmują czas stały $O(1)$.
- Pojemność listy jest nieograniczona.
- Potrzebujemy trzymać wskaźnik na początek i na koniec listy (ewentualnie lista jest cykliczna i trzymamy tylko wskaźnik na koniec listy).



Tablice

- Spójny obszar pamięci podzielony na komórki.
- Komórki w tablicy są numerowane liczbami całkowitymi począwszy od 0.
- Zaleta: czas dostępu do każdej komórki w tablicy jest stały $O(1)$.
- Wada: tablica jest strukturą nierozszerzalną.
- Zastosowanie: można z góry określić ilość danych; jest mało operacji wstawiania albo usuwania danych ze środka tablicy.

Tablice dynamiczne

- Tablica dynamiczna (ang. resizable array) dopasowuje swój rozmiar do rozmiaru danych tak, aby proporcja wykorzystanych komórek pamięci w stosunku do pojemności tablicy była ograniczona przez pewne stałe.
- Operacje na tablicy dynamicznej:
 - Dostęp do slotów w obrębie zapełnionych.
 - Dodanie nowej komórki na końcu tablicy (może się to wiązać z utworzeniem nowej dwukrotnie większej tablicy i z przepisaniem danych).
 - Usunięcie komórki z końca tablicy (może się to wiązać z utworzeniem nowej dwukrotnie mniejszej tablicy i z przepisaniem danych).

Tablice dynamiczne

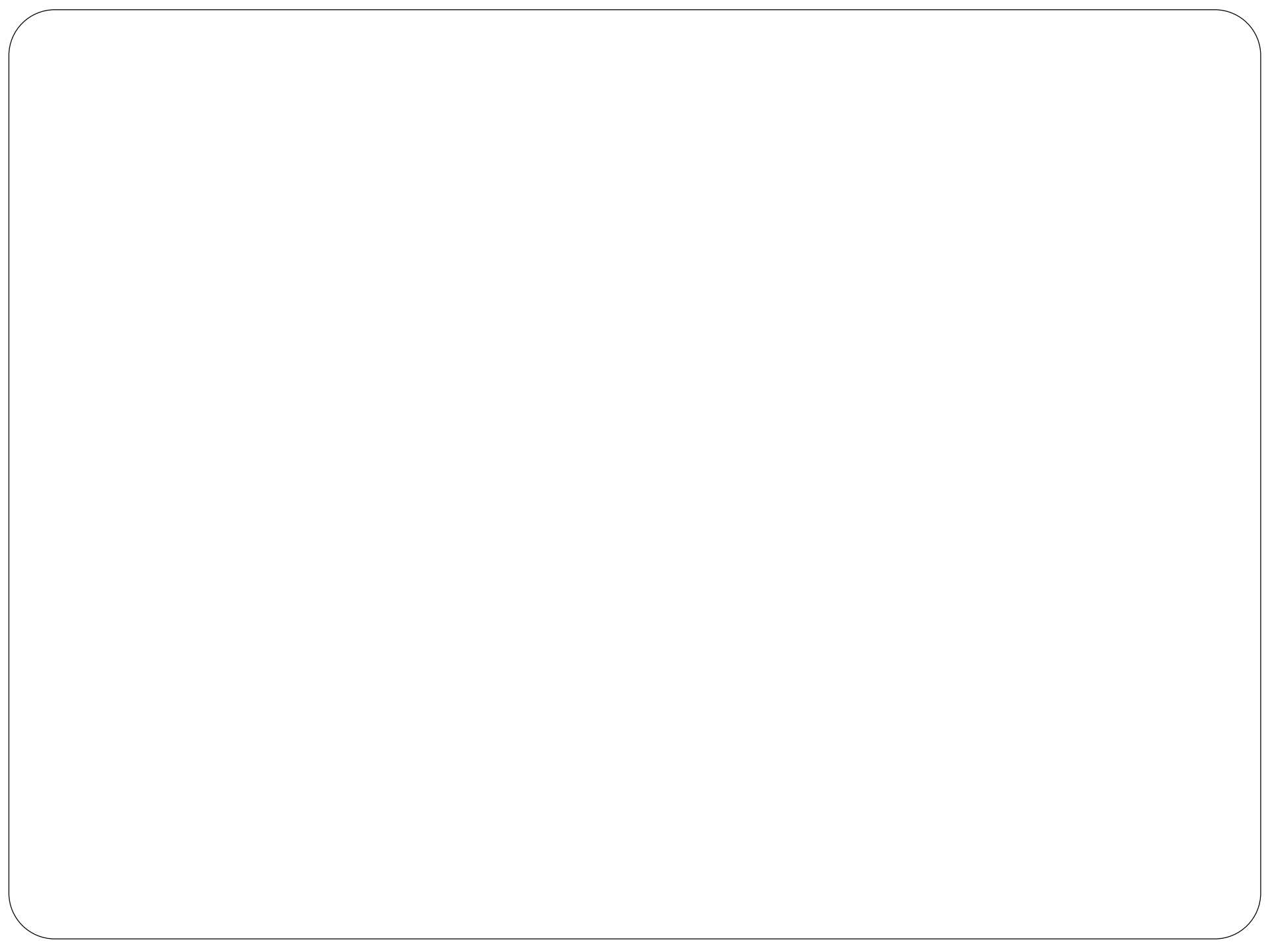
- Idea struktury:
 - Trzymamy tablicę z dodatkowymi parametrami: pojemność (liczba wszystkich slotów w tablicy) i wypełnienie (liczba wykorzystanych slotów w tablicy).
 - Gdy tablica się zapełni w 100%, powiększamy ją dwukrotnie.
 - Gdy wypełnienie tablicy spadnie poniżej 25%, pomniejszamy ją dwukrotnie.

Tablice dynamiczne

- Czasy operacji na tablicy dynamicznej:
 - Dostęp do zapełnionych slotów $O(1)$
 - Dodanie nowej komórki na końcu tablicy $O(n)$.
 - Usunięcie komórki z końca tablicy $O(n)$.
 - Uwaga: przeważnie dodanie/usunięcie komórki wymaga czasu $O(1)$.
- Analiza zamortyzowana:
 - Za pomocą metody księgowania jednostek kredytowych – każda operacja dodawania/usuwania elementu do/z tablicy pozostawia 2 kredyty czasowe do wykorzystania przy przepisywaniu tablicy.
 - Zamortyzowany koszt każdej pojedynczej operacji na tablicy dynamicznej jest stały $O(1)$ i wynosi 3 jednostki czasowe.

Tablice dynamiczne

- Zastosowanie: nie można z góry określić rozmiaru danych ale potrzebujemy szybkiego dostępu do każdej komórki w tablicy.
- Modyfikacje: tablica dynamiczna może się rozszerzać w obu kierunkach.
- Realizacja tej struktury w bibliotece standardowej C++: `vector<>`, `deque<>`.



Zadania i problemy

- Jak zasymulować działanie kolejki mając do dyspozycji tylko stosy?
- Jak zmodyfikować działanie tablicy dynamicznej, aby każda operacja powiększania i pomniejszania tablicy o jeden element działała w pesymistycznym czasie stałym $O(1)$?