Wrocław
University
of Science
and Technology

Struktury danych

Wykład 4

Kolejki

dr inż. Jarosław Rudy

HR EXCELLENCE IN RESEARCH

Kolejki (1)

- Kolejki (queue) to ADT, w którym dodawanie i usuwanie elementów jest ze sobą powiązane tj. usuwany element jest określony przez cechy elementów dodanych do tej pory (ich kolejność, priorytet itp.).
  - W praktyce kolejki utrzymują pewien (najczęściej liniowy) porządek elementów.
  - Wtedy dodawanie/usuwanie przekłada się na dodawane/usuwanie na którymś lub obu końcach kolejki.
- Podobnie jak dla listy elementy mogą się powtarzać i być różnego typu.
- Kolejki generalnie mają zmienną długość, ale rozważa się też przypadki o stałym rozmiarze.

Kolejki (2)

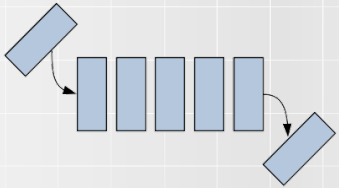
Typowe operacje na kolejkach:

- Dodanie elementu.
- Usunięcie elemenu.
- Sprawdzenie czy kolejka jest pusta (ewentualnie zwrócenie rozmiaru).
- Podgląd elementu do usunięcia (peek), ale bez usuwania.

Aby dostać się do elementu trzeciego, najpierw należy zdjąć elementy pierwszy i drugi! Kolejki zasadniczo nie służą do dostępu do dowolnego elementu, wyszukiwania czy przeglądania.

Kolejka FIFO (1)

- Kolejka działająca wg zasady First-In First-Out (elementy dodany jako pierwszy jest usuwany jako pierwszy).
- Operacja dodawania nazywa się enqueue, elementy dodawane są do „końca” (back, tail).
- Operacja zdejmowania nazywa się dequeue, elementy zdejmowane są z „początku” (front, head).
- Bez dodatkowego kontekstu „kolejka” zwykle oznacza „kolejka FIFO”.



Kolejka FIFO (2)

Implementacja z użyciem listy wiązanej jednokierunkowej (bez tail):

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addBack(e)	$O(n)$
dequeue()	removeFront()	$O(1)$
empty()	empty()	$O(1)$
peek()	get(0)	$O(1)$

Albo:

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addFront(e)	$O(1)$
dequeue()	removeBack()	$O(n)$
empty()	empty()	$O(1)$
peek()	get($n-1$)	$O(n)$

Kolejka FIFO (3)

Implementacja z użyciem listy wiązanej jednokierunkowej (z tail) lub dwukierunkowej:

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addBack(e)/addFront(e)	$O(1)$
dequeue()	removeFront()/removeBack()	$O(1)$
empty()	empty()	$O(1)$
peek()	get(0)/get($n-1$)	$O(1)$

Implementacja jest wydajna, ale współdzieli wady listy wiązanej (zajętość pamięci $2n + O(1)$, słabsza współpraca z pamięcią podręczną).

Kolejka FIFO (4)

Implementacja z użyciem tablicy dynamicznej:

Operacja kolejki	Operacja tablicy	Czas pesymistyczny
enqueue(e)	addBack(e)	$O(1)$ (amortyzowany)
dequeue()	removeFront()	$O(1)$ (amortyzowany)
empty()	empty()	$O(1)$
peek()	get(0)	$O(1)$

Albo:

Operacja kolejki	Operacja tablicy	Czas pesymistyczny
enqueue(e)	addFront(e)	$O(1)$ (amortyzowany)
dequeue()	removeBack()	$O(1)$ (amortyzowany)
empty()	empty()	$O(1)$
peek()	get($n-1$)	$O(1)$

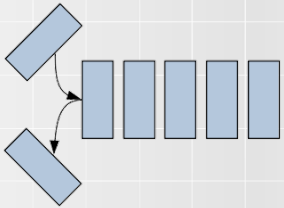
Kolejka FIFO (5)

Niektóre zastosowania:

- Bufory.
- Przechowywanie żądań obsługiwanych w kolejności zgłoszeń.
- Modelowanie kolejek sklepowych.
- Przegląd wszerz drzewa.
- Potoki w unixie.
- Komunikacja strumieniowa (np. TCP).

Stos (1)

- W stosie (ADT) element dodany najpóźniej jest zdejmowany jako pierwszy. Stos jest więc kolejką LIFO (Last-In Fist-Out).
- Elementy są więc dodawane i usuwane z tego samego końca (szczytu).
- Operacja dodawania nazywa się push().
- Operacja zdejmowania nazywa się pop().
- Operacja peek często nazywa się top().



Stos (2)

Implementacja z użyciem listy wiązanej jednokierunkowej (bez tail):

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addBack(e)	$O(n)$
dequeue()	removeBack()	$O(n)$
empty()	empty()	$O(1)$
top()	get($n-1$)	$O(n)$

Albo:

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addFront(e)	$O(1)$
dequeue()	removeFront()	$O(1)$
empty()	empty()	$O(1)$
top()	get(0)	$O(1)$

Stos (3)

Implementacja z użyciem listy wiązanej jednokierunkowej (z tail) lub dwukierunkowej:

Operacja kolejki	Operacja listy	Czas pesymistyczny
enqueue(e)	addFront(e)/addBack(e)	$O(1)$
dequeue()	removeFront()/removeBack()	$O(1)$
empty()	empty()	$O(1)$
top()	get(0)/get($n-1$)	$O(1)$

Stos (4)

Implementacja z użyciem tablicy dynamicznej:

Operacja kolejki	Operacja tablicy	Czas pesymistyczny
enqueue(e)	addBack(e)	$O(1)$ (amortyzowany)
dequeue()	removeBack()	$O(1)$ (amortyzowany)
empty()	empty()	$O(1)$
top()	get($n-1$)	$O(1)$

Albo:

Operacja kolejki	Operacja tablicy	Czas pesymistyczny
enqueue(e)	addFront(e)	$O(1)$ (amortyzowany)
dequeue()	removeFront()	$O(1)$ (amortyzowany)
empty()	empty()	$O(1)$
top()	get(0)	$O(1)$

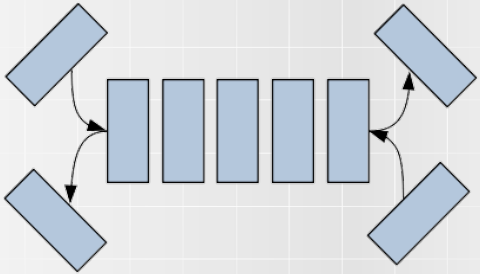
Stos (5)

Niektóre zastosowania:

- Parsowanie wyrażeń (np. arytmetycznych) zapisanych w odwrotnej notacji polskiej.
- Przegląd w głąb drzewa.
- Algorytmy z nawrotami (backtracking).
- Stos programowy.

Kolejka dwukierunkowa (1)

- Kolejka, w której można dodawać i usuwać elementy na obu końcach.
- Double-ended que = deque (czyt. „dek").
  - Nie mylić z operacją dequeue (czyt. „dikju") kolejki FIFO!



Kolejka dwukierunkowa (2)

- Stos i kolejka FIFO mogą być traktowane jako uszczegółowienie deque.
- Deque może być traktowana jako uszczegółowienie listy.

Operacja	List	Deque	FIFO	Stack
addFront(e)	✓	✓	✗	✓
removeFront()	✓	✓	✓	✓
addBack(e)	✓	✓	✓	✗
removeBack()	✓	✓	✗	✗
add(e,i)	✓	✗	✗	✗
remove(i)	✓	✗	✗	✗

Kolejka dwukierunkowa (3)

Implementacja z użyciem listy wiązanej jednokierunkowej (bez tail):

Operacja kolejki	Operacja listy	Czas pesymistyczny
addFront(e)	addFront(e)	$O(1)$
addBack(e)	addBack(e)	$O(n)$
removeFront()	removeFront()	$O(1)$
removeBack()	removeBack()	$O(n)$
empty()	empty()	$O(1)$
front()	get(0)	$O(1)$
back()	get($n-1$)	$O(n)$

Kolejka dwukierunkowa (4)

Implementacja z użyciem listy wiązanej jednokierunkowej (z tail) lub dwukierunkowej:

Operacja kolejki	Operacja listy	Czas pesymistyczny
addFront(e)	addFront(e)	$O(1)$
addBack(e)	addBack(e)	$O(1)$
removeFront()	removeFront()	$O(1)$
removeBack()	removeBack()	$O(1)$
empty()	empty()	$O(1)$
front()	get(0)	$O(1)$
back()	get($n-1$)	$O(1)$

Kolejka dwukierunkowa (5)

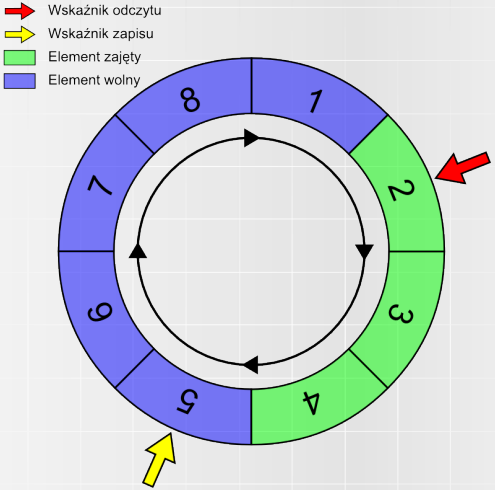
Implementacja z użyciem tablicy dynamicznej:

Operacja kolejki	Operacja tablicy	Czas pesymistyczny
addFront(e)	addFront(e)	$O(1)$ (amortyzowany)
addBack(e)	addBack(e)	$O(1)$ (amortyzowany)
removeFront()	removeFront()	$O(1)$ (amortyzowany)
removeBack()	removeBack()	$O(1)$ (amortyzowany)
empty()	empty()	$O(1)$
front()	get(0)	$O(1)$
back()	get($n-1$)	$O(1)$

Kolejka cykliczna (1)

- Kolejka (bufor) cykliczny jest strukturą danych, zwykle o stałym rozmiarze.
- Najczęściej działa w charakterze FIFO.
- Dwa wskaźniki na aktualny początek (odczyt) i koniec (zapis) kolejki.
- Pozycje wskaźników aktualizowane przy operacjach dodawania i zdejmowania.
- Stare dane są zapisywane przez nowe.
- Teoretycznie może służyć do przesyłu dowolnie dużych danych
  - Gdy jeden wksaźnik dogoni drugi, operacja możliwa dopiero po przesunięciu drugiego wskaźnika.

Kolejka cykliczna (2)



Kolejka cykliczna (3)

- Prostota, oszczędność miejsca, szybki dostęp.
- Ograniczony rozmiar.
  - Dynamiczne bufory cykliczne.
- Stosowane np. do implementacji bufora klawiatury.
- Przy dobrej implementacji czas operacji wynosi $O(1)$.
  - Tablica dynamiczna wymaga „ręcznego” zawijania indeksów.
  - Lista wiązana działa dzięki pamiętaniu wskaźników.

Kolejka priorytetowa (1)

- Kolejka w której każdy element ma przypisany priorytet liczbowy.
  - Ogólniej, elementy są parą klucz-wartość, gdzie na kluczach da się określić relację maksimum (minimum). Klucze muszą mieć więc częściowy porządek.
  - Priorytetem może być po prostu wartość elementu.
- O „kolejności" kolejki decyduje priorytet elementów:
  - Dodawanie elementu e o priorytecie p dodaje go (jakoś) do kolekcji.
  - Zdejmowanie zawsze zdejmuje element o największym (najmniejszym) priorytecie.
- Może istnieć wiele elementów o tym samym priorytecie. Różne strategie:
  - Stabilność – gwarancja zdejmownia takich elementów w kolejności ich dodawania (FIFO).
  - Niestabilne – brak takiej gwarancji.

Kolejka priorytetowa (2)

Kolejka priorytetowa typu max:

- insert(e,p) – dodanie elementu e o priorytecie p.
- extract-max() – usunięcie i zwrócenie elementu o największym priorytecie.
- find-max() – zwrócenie (podejrzenie) elementu o największym priorytecie.
- modify-key(e,p) – zmiana priorytetu elementu e na p. Można podzielić na operacje decrease-key oraz increase-key.

Kolejka priorytetowa typu min jest analogiczna (extract-min zwraca element o najmniejszym priorytecie itp).

Kolejka priorytetowa (3)

- Pierwszą (naiwną) implementacją kolejki priorytetowej jest wykorzystanie listy, czyli użycie wprost tablicy dynamicznej lub listy wiązanej.
- Zakładamy, że struktury posiadają odpowiednie usprawnienia, inaczej przedstawione dalej złożoności mogą nie być zawsze spełnione.
- Implementacja z użyciem listy wiązanej wykorzystuje więcej pamięci i może gorzej współpracować z pamięcią podręczną procesora.
- Przedstawiona implementacja dotyczy kolejki priorytetowej typu max. Implementacja kolejki typu min jest analogiczna.

Kolejka priorytetowa (4)

- Dodajemy elementy na koniec jak w zwykłej kolejce FIFO. Przy zdejmowaniu należy znaleźć największy element.
- Operacje:
  - insert(e,p) – za pomocą addBack(e), czas $O(1)$ (zwykły lub amortyzowany).
  - extract-max() – za pomocą przeszukania listy, zwrócenia i usunięcia znalezionego elementu, czas $O(n)$.
  - peek() – analogicznie do extract-max(), ale bez usuwania, czas $O(n)$.
  - modify-key(e,p) – znalezienie elementu i modyfikacja jego priorytetu, czas $O(n)$.

Kolejka priorytetowa (5)

- Możemy również odwrócić koncepcję – dodajemy elementy od razu w potrzebne miejsce (jak insert sort), wtedy lista jest posortowana i największy elemnt będzie zawsze na początku.
- Operacje:
  - insert(e,p) – za pomocą wyszukania odpowiedniego miejsca i wstawienia, czas $O(n)$.
  - extract-max() – zwracamy i usuwamy pierwszy element, czas $O(1)$ (amortyzowany lub nie).
  - peek() – analogicznie do extract-max(), ale bez usuwania, czas $O(1)$ (amortyzowany lub nie).
  - modify-key(e,p) – znalezienie elementu, modyfikacja jego priorytetu i przeniesienie elementu w odpowiednie miejsce, czas $O(n)$.