Fakultas Informatika
School of Computing
Telkom University
Think Big! Go Forward!

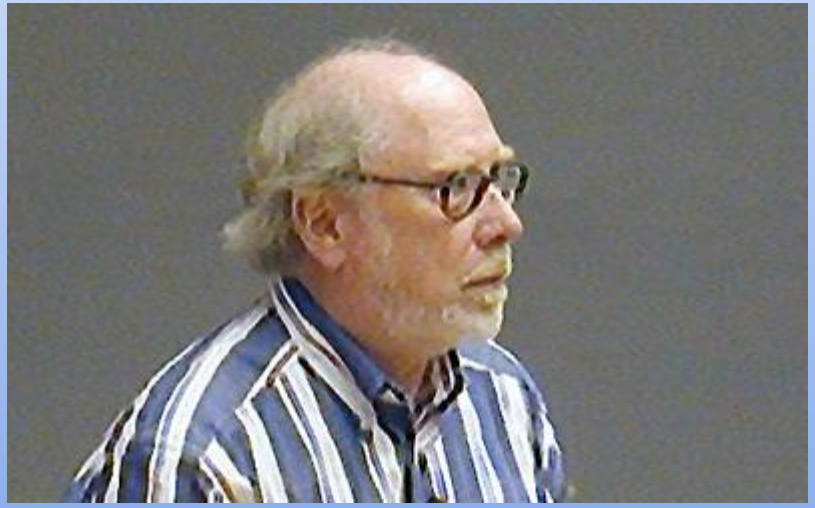
Telkom University

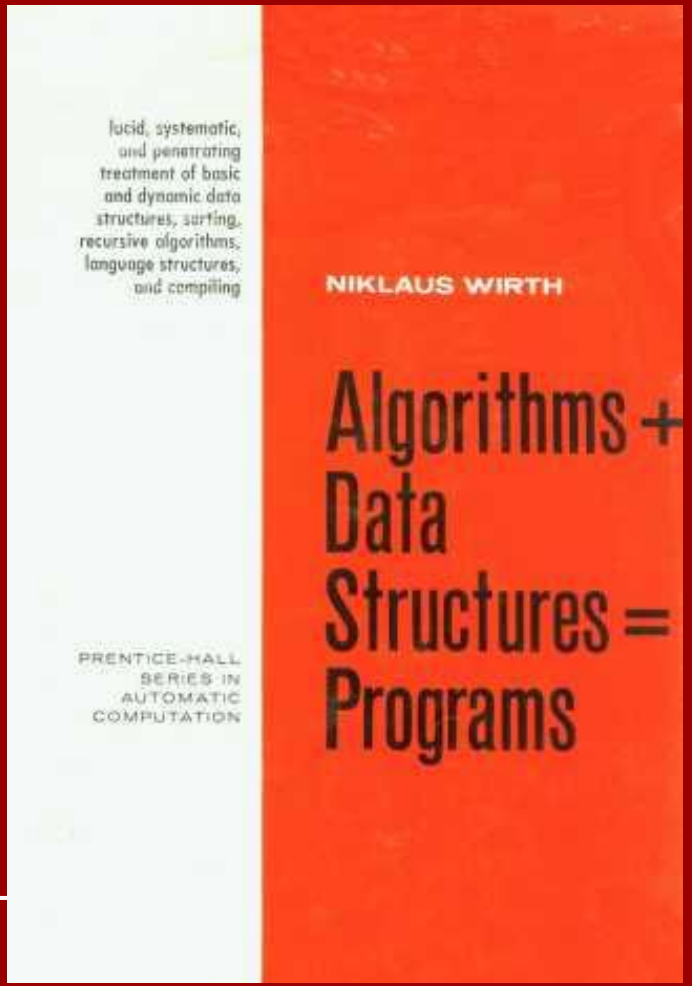
ANALISIS ALGORITMA

Week 05: Pengantar Struktur Data

PROGRAM PASCA SARJANA INFORMATIKA
FAKULTAS TEKNIK INFORMATIKA
UNIVERSITAS TELKOM

2022/2023




lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling
NIKLAUS WIRTH
Algorithms + Data Structures = Programs
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

Data Structures, Pengantar

Stack dan Queue: Operasi Primitif

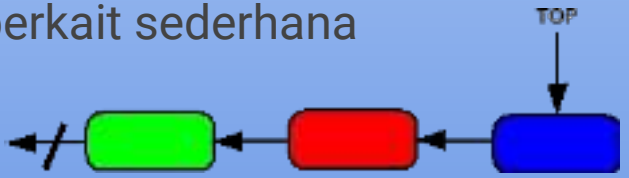
- **Clear(Q)** : siapkan queue Q agar operasional
- **Enqueue(Q, v)**: simpan suatu nilai v kedalam queue Q
  Penyimpanan sedemikian rupa agar saat pengambilan sesuai prioritas
- **Requeue(Q, v)**: simpan atau sesuaikan posisi nilai v dalam queue Q
  Operasi mirip seperti **enqueue(Q,v)**, untuk membantu proses **dequeue(Q)**
- **Dequeue(Q)**: mengambil data dari queue Q
  Operasi utama, mengembalikan nilai dan menghapus dari queue
- **Empty(Q)**: memeriksa apakah tidak ada data dalam queue Q
  Setelah op **clear(Q)** fungsi **empty(Q)** harus memberikan nilai **True**

Stack and Queue: Operasi Stack

- Push(Q, v) ≡ Enqueue(Q, v)
- Pop(Q) ≡ Dequeue(v) : ambil data yang terakhir disimpan
- Swap(Q) : tukar data diantara dua data terakhir disimpan
- Pop(Q,i) : ambil data ke-i menurut urutan kedatangan

Alternatif implementasi:

- Array sederhana stack[1..N] dimana indeks 1 paling bawah, dan indeks top menunjuk pada data terbaru.
- List berkait sederhana




```
type Stack struct
    data[1..N]
    top : index
endstruct
```

Stack and Queue: Kompleksitas Op. Stack

```
proc clear(Stack Q)
    Q.top = 0
endproc
```

```
func empty(Stack Q) → Bool
    return Q.top < 1
endproc
```

```
proc swap(Stack Q)
    if Q.top > 1 then
        Q.data[Q.top] ⇄ Q.data[Q.top-1]
    endif
endproc
```

```
func pop(Stack Q) → Data
    if not empty(Q) then
        value = Q.data[Q.top]
        Q.top--
    endif
    return value
endfunc
```

```
proc push(Stack Q, v)
    if Q.top < N then
        Q.top++
        Q.data[Q.top] = v
    endif
endproc
```

Stack and Queue: Kompleksitas Op. Stack...

```
func pop(Stack Q, i) → Data
    if Q.top >= i > 0 then
        value = Q.data[i]
        while i < Q.top do
            Q.data[i] = Q.data[i+1]
            i++
        endwhile
        Q.top--
    endif
    return value
endfunc
```

```
func peek(Stack Q) → Data
    if Q.top > 0 then
        value = Q.data[Q.top]
    endif
    return value
endfunc
```

Stack and Queue: Kompleksitas Op Stack … …

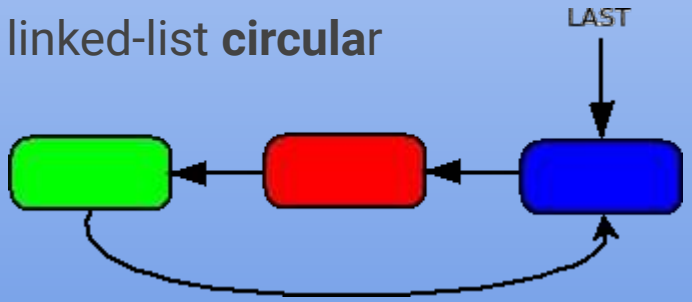
- Semua operasi Stack memerlukan O(1),
- Kecuali operasi pop(Q,i) memerlukan O(n) on worst case
- Array dan linked list akan mirip;
  - Array lebih efisien, tetapi kapasitas terbatas/dibatasi
  - Linked list fleksibel, tetapi alokasi space membutuhkan overhead untuk pointer
- Karakteristik Queue (FIFO), apakah mirip?

Stack and Queue: Operasi Queue FIFO

Implementasi:

- Pointer ke data **pertama**
- Menyimpan **jumlah** data
- Dengan array sederhana queue[1..N]
  → Bagaimana mendaur-ulang space setelah operasi dequeue?
- Menggunakan array **circular**
- Atau linked-list **circular**

```
type Queue struct
    data[1..N]
    first, num
endstruct
```



Stack and Queue: Kompleksitas Op Queue

```
proc clear(Queue Q)
    Q.first = 1
    Q.num = 0
endproc
```

```
func dequeue(Queue Q) → Data
    if not empty(Q) then
        value = Q.data[Q.first]
        Q.first = (Q.first mod N)+1
        Q.num--
    endif
    return value
endfunc
```

```
func empty(Queue Q) → Bool
    return Q.num <= 0
endproc
```

```
proc enqueue(Queue Q, v)
    if not full(Q) then
        p = ((Q.first+Q.num-1) mod N) + 1
        Q.data[p] = v
        Q.num++
    endif
endproc
```

```
func full(Queue Q) → Bool
    return Q.num >= N
endproc
```

Stack and Queue: Kompleksitas Op Queue

```
func dequeue(Queue Q, i) → Data
    if 1 <= i <= Q.num then
        ipos = ((i+Q.first-2) mod N) +1
        value = Q.data[ipos]
        while i < Q.num do
            ipos1 = ipos mod N + 1
            Q.data[ipos] = Q.data[ipos1]
            ipos = ipos1
            i++
        endwhile
        Q.num--
    endif
    return value
endfunc
```

Stack and Queue: Ops Queue FIFO

- Perlu usaha lebih untuk men-daur-ulang memori
- Rancangan algoritma harus lebih hati-hati untuk mendapatkan op. dg. O(1),
- Juga mirip, dequeue(Q, i) tetap perlu O(n) untuk menggeser data lainnya

- Apakah kita bisa mendapatkan kompleksitas lebih baik untuk semua op?
- Dapatkah kita menerapkan prioritas lainnya untuk ops. dequeue/pop?
  - Jangan hanya data pertama atau data terakhir
  - Tetapi dengan konsep prioritas lain!

- Perlu menyatakan prioritas secara eksplisit
- Bisa menggunakan struktur **heap**!

```
type Heap struct
    data[1..N]
    num
endstruct
```

Stack and Queue: Kompleksitas Priority Queue

```
proc clear(Heap Q)
    Q.num = 0
endproc
```

```
func empty(Queue Q) → Bool
    return Q.num == 0
endproc
```

```
proc enqueue(Queue Q, v)
    if not full(Q) then
        Q.num++
        Q.data[Q.num] = v
        fixheapUp(Q[1..Q.num], Q.num)
    endif
endproc
```

```
func dequeue(Queue Q) → Data
    if not empty(Q) then
        value = Q.data[1]
        Q.data[1] = Q.data[Q.num]
        Q.num--
        fixheapDown(Q[1..Q.num],1)
    endif
    return value
endfunc
```

```
func full(Queue Q) → Bool
    return Q.num == N
endproc
```

Stack and Queue: Priority Queue, Complexities?

```
proc requeue(Queue Q, i, v)
    if 0 < i <= Q.num then
        if Q.data[i] < v then
            Q.data[i] = v
            fixheapUp(Q[1..Q.num],i)
        else
            Q.data[i] = v
            fixheapDown(Q[1..Q.num],i)
    endif
endproc
```

```
func dequeue(Queue Q, i) → Data
    if 0 < i <= Q.num then
        Q.data[i] = Q.data[Q.num]
        Q.num--
        fixheapDown(Q[1..Q.num],i)
    endif
    return value
endfunc
```

Stack and Queue: Kompleksitas Priority Queue

- clear(), empty(), full() ops membutuhkan O(1)
- Worst case ops enqueue() dan dequeue() merayap setinggi pohon, O($\lg n$)
- dequeue(Q,i) dengan i sebuah node dalam tree, rerata O(1), worst case O($\lg n$)
- requeue(Q,i,v) mirip, rerata O(1) dan worst case O(***lg***n)
- … Jika diberikan 2 atau lebih queue… dapatkah digabungkan menjadi satu queue?
- Pasti bisa, tetapi seberapa efisien algoritma penggabungan tersebut?

Operation	(*)Simple array	PriorityHeap	Binomial Heap
`H:=MakeQueue()`	O(1)	O(1)	O(1)
`Insert(H,x)`	O(1)	O(*lg* n)	O(1)*
`x:=ExtractMax(H)`	O(n)	O(*lg* n)	O(*lg* n)
`UpdateKey(H,x,k)`	O(1)/O(n)	O(*lg* n)	O(*lg* n)
`b:=IsEmpty(H)`	O(1)	O(1)	O(1)
`x:=FindMax(H)`	O(n)	O(1)	O(1)
`Delete(H,x)`	O(n)	O(*lg* n)	O(*lg* n)
`H:=Splice(`H_1H_2`)`	O(min(n_1,n_2))	O(min(n_1,n_2))	O(1)*

Heap of Heaps

- b-heap
- fat heap
- 2-3 heap
- leaf heap
- thin heap
- skew heap
- splay heap
- weak heap
- leftist heap
- quake heap
- pairing heap
- violation heap
- run-relaxed heap
- rank-pairing heap
- skew-pairing heap
- rank-relaxed heap
- fibonacci heap
- lazy fibonacci heap
- brodal queue
- strict fibonacci heap

Struktur Data

Struktur Data

- Data distrukturkan agar komputasi menjadi lebih mudah/lebih efisien
- Beberapa struktur yang umum:
  - Array dan Record
  - Linked List
  - Tree dan Graph
  - Hash dan data mapping
- Operasi primitif yang diharapkan tersedia:
  - Insert, Append, Add, ...
  - Delete, Remove, Erase, ...
  - Search dan Find extremes, ...
  - Agregate operations: Sum, Count, List, ...

Kompleksitas Beberapa Struktur Sederhana

Primitives	U-Array	O-Array	Linked List	O Linked List	D Linked List	Circular List	D Circular List
Search	O(n)	O(lg n)	O(n)	O(n)	O(n)		
FindMax	O(n)	O(1)	O(n)	O(n) / O(1)	O(1)		
Insert	O(1)	O(n)	O(1)	O(n)	O(n) / O(1)		
Delete	O(n)	O(n)	O(n)	O(n)	O(1)		

Notes:

- U-Array = unordered array
- O-Array = ordered array
- O Linked List = ordered linked list
- D Linked List = doubly linked list
- D Circular List = doubly circular linked list