Priority Queues

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 6

Dealing with priorities

Job scheduler

- A job scheduler maintains a list of pending jobs with their priorities

Dealing with priorities

Job scheduler

- A job scheduler maintains a list of pending jobs with their priorities
- When the processor is free, the scheduler picks out the job with maximum priority in the list and schedules it

Dealing with priorities

Job scheduler

- A job scheduler maintains a list of pending jobs with their priorities
- When the processor is free, the scheduler picks out the job with maximum priority in the list and schedules it
- New jobs may join the list at any time

Dealing with priorities

Job scheduler

- A job scheduler maintains a list of pending jobs with their priorities
- When the processor is free, the scheduler picks out the job with maximum priority in the list and schedules it
- New jobs may join the list at any time
- How should the scheduler maintain the list of pending jobs and their priorities?

Dealing with priorities

Job scheduler

- A job scheduler maintains a list of pending jobs with their priorities
- When the processor is free, the scheduler picks out the job with maximum priority in the list and schedules it
- New jobs may join the list at any time
- How should the scheduler maintain the list of pending jobs and their priorities?

Priority queue

- Need to maintain a collection of items with priorities to optimise the following operations
- `delete_max()`
  - Identify and remove item with highest priority
  - Need not be unique
- `insert()`
  - Add a new item to the collection

Implementing priority queues with one dimensional structures

- `delete_max()`
  - Identify and remove item with highest priority
  - Need not be unique
- `insert()`
  - Add a new item to the list

Implementing priority queues with one dimensional structures

- Unsorted list
  - `insert()` is $O(1)$
  - `delete_max()` is $O(n)$

- `delete_max()`
  - Identify and remove item with highest priority
  - Need not be unique
- `insert()`
  - Add a new item to the list

Implementing priority queues with one dimensional structures

- Unsorted list
  - `insert()` is $O(1)$
  - `delete_max()` is $O(n)$
- Sorted list
  - `delete_max()` is $O(1)$
  - `insert()` is $O(n)$

- `delete_max()`
  - Identify and remove item with highest priority
  - Need not be unique
- `insert()`
  - Add a new item to the list

Implementing priority queues with one dimensional structures

- Unsorted list
  - `insert()` is $O(1)$
  - `delete_max()` is $O(n)$
- Sorted list
  - `delete_max()` is $O(1)$
  - `insert()` is $O(n)$
- Processing n items requires $O(n^2)$

- `delete_max()`
  - Identify and remove item with highest priority
  - Need not be unique
- `insert()`
  - Add a new item to the list

Moving to two dimensions

First attempt

- Assume N processes enter/leave the queue

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

Moving to two dimensions

First attempt

- Assume N processes enter/leave the queue
- Maintain a $\sqrt{N} \times \sqrt{N}$ array

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

Moving to two dimensions

First attempt

- Assume N processes enter/leave the queue
- Maintain a $\sqrt{N} \times \sqrt{N}$ array
- Each row is in sorted order

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

insert()

- Keep track of the size of each row

$N = 25$

3	19	23	35	58		5
12	17	25	43	67		5
10	13	20				3
11	16	28	49			4
6	14					2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

5
5
3
4
2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

5
5
3
4
2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

15

						Size
3	19	23	35	58		5
12	17	25	43	67		5
10	13	20				3
11	16	28	49			4
6	14					2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

	3	19	23	35	58	5
15	12	17	25	43	67	5
	10	13	20			3
	11	16	28	49		4
	6	14				2

`insert()`

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

	3	19	23	35	58	5
	12	17	25	43	67	5
15	10	13	20			3
	11	16	28	49		4
	6	14				2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
3
4
2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
4
4
2

insert()

- Keep track of the size of each row
- Insert into the first row that has space
  - Use size of row to determine
- Insert 15
- Takes time $O(\sqrt{N})$
  - Scan size column to locate row to insert, $O(\sqrt{N})$
  - Insert into the first row with free space, $O(\sqrt{N})$

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
4
4
2

delete_max()

- Maximum in each row is the last element

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
4
4
2

delete_max()

- Maximum in each row is the last element
- Position is available through size column

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
4
4
2

delete_max()

- Maximum in each row is the last element
- Position is available through size column
- Identify the maximum amongst these

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	15	20	
11	16	28	49	
6	14			

5
5
4
4
2

delete_max()

- Maximum in each row is the last element
- Position is available through size column
- Identify the maximum amongst these
- Delete it

$N = 25$

3	19	23	35	58
12	17	25	43	
10	13	15	20	
11	16	28	49	
6	14			

5
4
4
4
2

delete_max()

- Maximum in each row is the last element
- Position is available through size column
- Identify the maximum amongst these
- Delete it
- Again $O(\sqrt{N})$
  - Find the maximum among last entries, $O(\sqrt{N})$
  - Delete it, $O(1)$

$N = 25$

						size
3	19	23	35	58		5
12	17	25	43			4
10	13	15	20			4
11	16	28	49			4
6	14					2

Summary

- 2D $\sqrt{N} \times \sqrt{N}$ array with sorted rows
  - `insert()` is $O(\sqrt{N})$
  - `delete_max()` is $O(\sqrt{N})$
  - Processing N items is $O(N\sqrt{N})$

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

Summary

- 2D $\sqrt{N} \times \sqrt{N}$ array with sorted rows
  - `insert()` is $O(\sqrt{N})$
  - `delete_max()` is $O(\sqrt{N})$
  - Processing N items is $O(N\sqrt{N})$
- Can we do better?

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

Summary

- 2D $\sqrt{N} \times \sqrt{N}$ array with sorted rows
  - `insert()` is $O(\sqrt{N})$
  - `delete_max()` is $O(\sqrt{N})$
  - Processing N items is $O(N\sqrt{N})$
- Can we do better?
- Maintain a special binary tree — heap
  - Height $O(\log N)$
  - `insert()` is $O(\log N)$
  - `delete_max()` is $O(\log N)$
  - Processing N items is $O(N \log N)$

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			

Summary

- 2D $\sqrt{N} \times \sqrt{N}$ array with sorted rows
  - `insert()` is $O(\sqrt{N})$
  - `delete_max()` is $O(\sqrt{N})$
  - Processing N items is $O(N\sqrt{N})$
- Can we do better?
- Maintain a special binary tree — heap
  - Height $O(\log N)$
  - `insert()` is $O(\log N)$
  - `delete_max()` is $O(\log N)$
  - Processing N items is $O(N \log N)$
- Flexible — need not fix N in advance

$N = 25$

3	19	23	35	58
12	17	25	43	67
10	13	20		
11	16	28	49	
6	14			