

CS010C

Lab8(No.8 Lab)

heap

Min-Heap Review

- See slides
 - CANVAS -> modules -> week4 -> Binary Heap Slides
- We have n nodes, from 0 to $n - 1$
 - Node i 's
 - Parent = $(i - 1)/2$
 - Left Child = $2i + 1$
 - Right Child = $2i + 2$
 - Insert
 - Put the new node at n -th position, then **trickle up**
 - (iteratively compare with parent)
 - Remove_min
 - Remove the root, place last item into the hole, then **trickle down**
 - (iteratively compare with the smaller child - remember 2 children)
- Build heap
 - $O(n)$, rather than $O(n\log n)$

```
// Trickle up
i = numNodes // i == size
while (i > 0 && h[(i-1)/2] > item)
    h[i] = h[(i-1)/2]
    i = (i-1)/2
h[i] = item
numNodes++
```

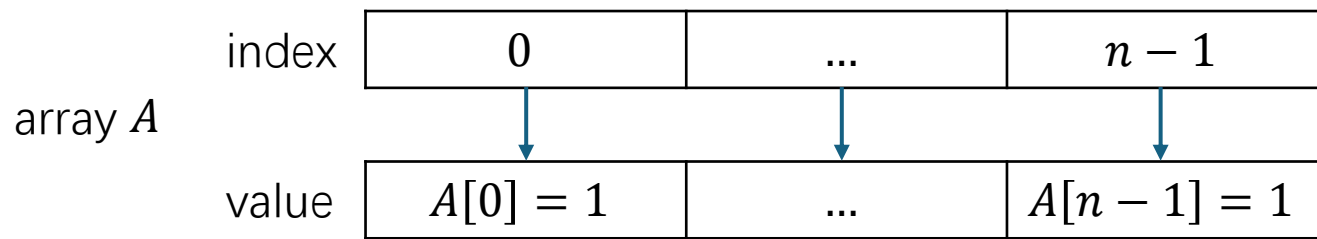
Remove highest priority item

- Makes a hole at the root
- Want to remain a complete tree, so attempt to place last item in the heap into the hole
 - If item can be placed in hole without violation of the heap property, then done
 - Otherwise, trickle down
 - **Pick the child with the highest priority - two children**

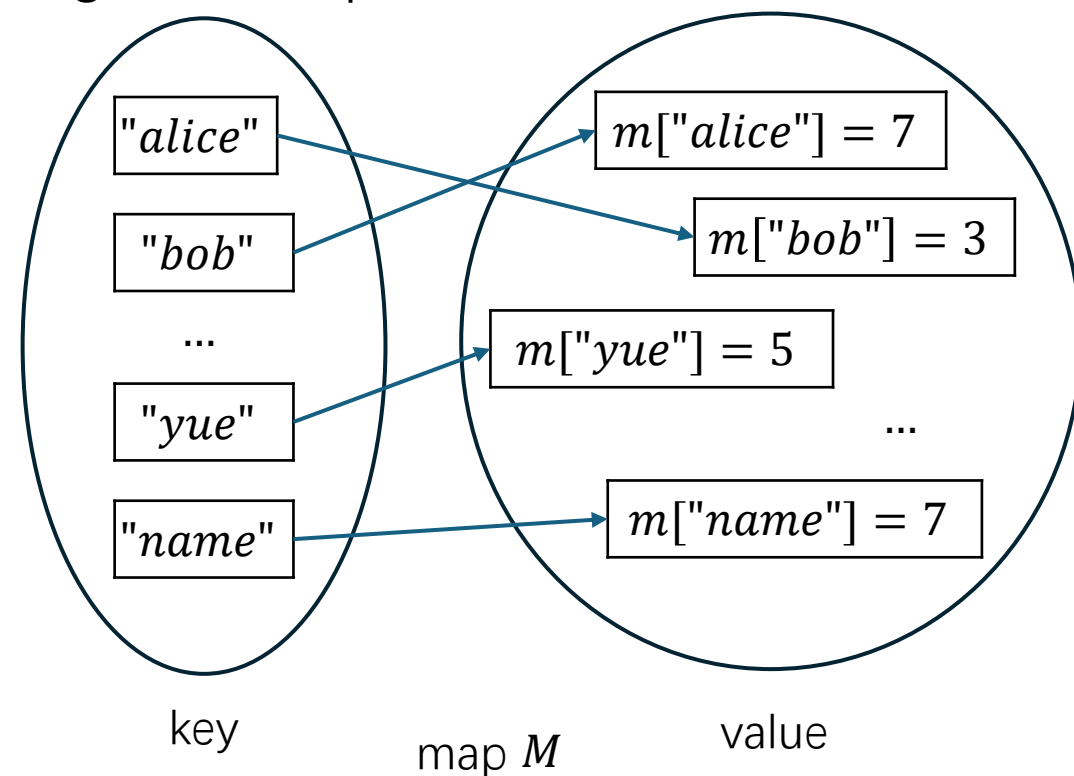
map

powerful than vector/array

map



- Review: what is an array? consists of **index-value pairs**
 - an *index*, must be integer
 - you can get the *value* at the corresponding *index*'s position
 - *index* is unique, *value* can be repeated
- A map: consists of **key-value pairs**
 - a *key*, can be anything
 - you can look up the *value* by the *key*
 - *key* is unique, *value* can be repeated



std::map example

```
#include <iostream>
#include <map>
#include <string>

int main ()
{
    std::map<char, std::string> mymap;

    mymap['a'] = "an element";
    mymap['b'] = "another element";
    mymap['c'] = mymap['b'];

    std::cout << "mymap['a'] is " << mymap['a'] << '\n';
    std::cout << "mymap['b'] is " << mymap['b'] << '\n';
    std::cout << "mymap['c'] is " << mymap['c'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";

    std::cout << "-----UPDATE mymap['c']-----\n";
    mymap['c'] = "something else";
    std::cout << "mymap['c'] is " << mymap['c'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";

    std::cout << "-----WHAT IF mymap['d'] IS NOT SET BUT IS ACCESSED-----\n";
    std::cout << "mymap['d'] is " << mymap['d'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";
    return 0;
}
```

Example:

https://github.com/SJZHZ/UCR_CS010C_25U/blob/main/demos/Lab7/map.cpp

```
> bash run.sh
mymap['a'] is an element
mymap['b'] is another element
mymap['c'] is another element
mymap now contains 3 elements.
-----UPDATE mymap['c']-----
mymap['c'] is something else
mymap now contains 3 elements.
-----WHAT IF mymap['d'] IS NOT SET BUT IS ACCESSED-----
mymap['d'] is
mymap now contains 4 elements.
```

🏠 ~ / UCR_CS010C_25U / demos / Lab7 🐱 main !2 ?3

priority queue

pq_zero.H overview

- Completed
 - “heap” vector stores <Item>s
 - “index” map stores <Item,int> pairs
 - “priority” map stores <Item,float> pairs
- Relatively complex
 - `void percolate_up(indx i);`
 - `void percolate_down(indx i);`
- Straightforward
 - Once you have completed the basic operations

```
private:
    vector<Item> heap; // The heap expands/shrinks to fit data
    typedef int indx; // index with heap
    map<Item,indx> index; // records each Item's place in heap
    map<Item,float> priority; // records each Item's priority
    void percolate_up( indx i );
    void percolate_down( indx i );
```

Always remember we have these 3 variables!

```
public:
    // These use the min-heap functions above.
    int size( ) const;
    bool empty( ) const;
    const Item& front( ) const;
    void pop( );
    void push( const Item& w, float prio );
};
```


heap

- element
 - only priority value
- Given i , how to
 - Find parent/child
 - Get priority value
 - Move to j
 - Swap with j

$$\begin{aligned}\text{Parent} &= (i - 1)/2 \\ \text{Left Child} &= 2i + 1 \\ \text{Right Child} &= 2i + 2\end{aligned}$$

$h[i]$

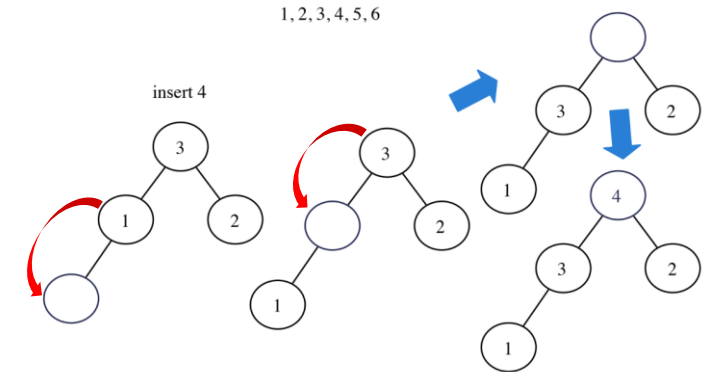
$h[i]$

$h[j] = h[i]$

$temp = h[i], h[i] = h[j], h[j] = temp$

- **percolate_up** pseudocode
 - While (not_root && parent_larger)
 - Move parent to child
 - Go upward to parent's position
 - Put the item to current position
 - Count++
- Likewise for **percolate_down**

Max Heap: insert

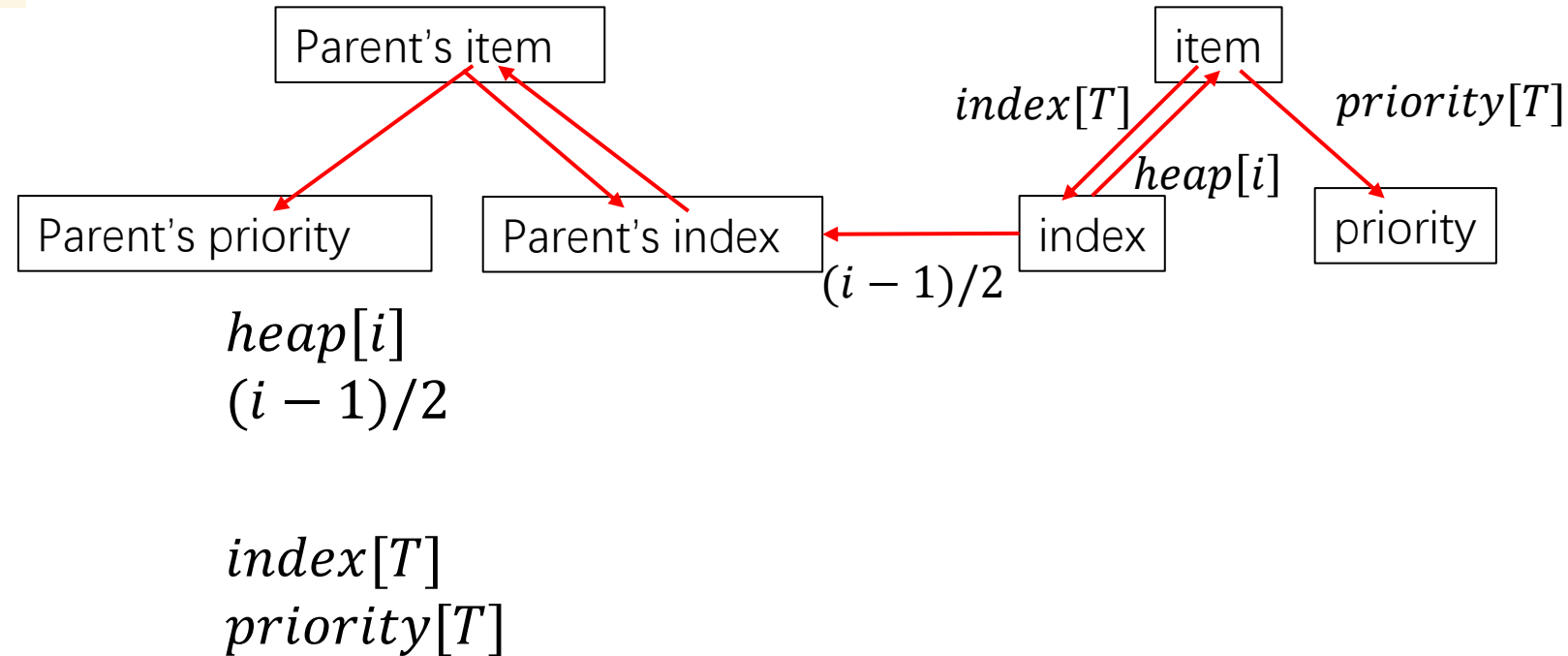


```
// Trickle up
i = numNodes // i == size
while (i > 0 && h[(i-1)/2] > item)
    h[i] = h[(i-1)/2]
    i = (i-1)/2
h[i] = item
numNodes++
```

```
vector<Item> heap; // The
map<Item,indx> index; //
map<Item,float> priority;
```

Data structure

- Given index i , how to
 - Get item
 - Get parent
- Given item T , how to
 - Get index
 - Get priority



- Puzzle
 - Given index i , how to get priority
 - Index \rightarrow item \rightarrow priority!
 - Given item T , how to get its parent
 - Item \rightarrow index \rightarrow parents's index!

priority[heap[i]]

heap[(index[T] - 1)/2]

Operation

```
vector<Item> heap; // The  
map<Item,indx> index; //  
map<Item,float> priority;
```

- create a node at index i
 - *heap[i] = new_item*
 - *index[new_item] = i*
- Puzzle: how to swap i and j nodes
- We have “heap”, “index”, “priority”
 - What will change?
 - “heap” & “index”
 - “priority”? The relationship between item & priority remains unchanged!
 - How will they change?
 - *swap(heap[i], heap[j]);* // swaps strings in heap
 - *swap(index[heap[i]], index[heap[j]]);* // updates string's position in heap

PQ

```
vector<Item> heap; // The  
map<Item,indx> index; //  
map<Item,float> priority;
```

- **percolate_up**

- Put the new item to last position
- While (not_root && parent_larger)
 - swap parent and child
 - Go upward to parent's position
- Count++

- Likewise for **percolate_down**

*heap[i] = new_item
index[new_item] = i*

*swap(heap[i], heap[j]);
swap(index[heap[i]], index[heap[j]]);*