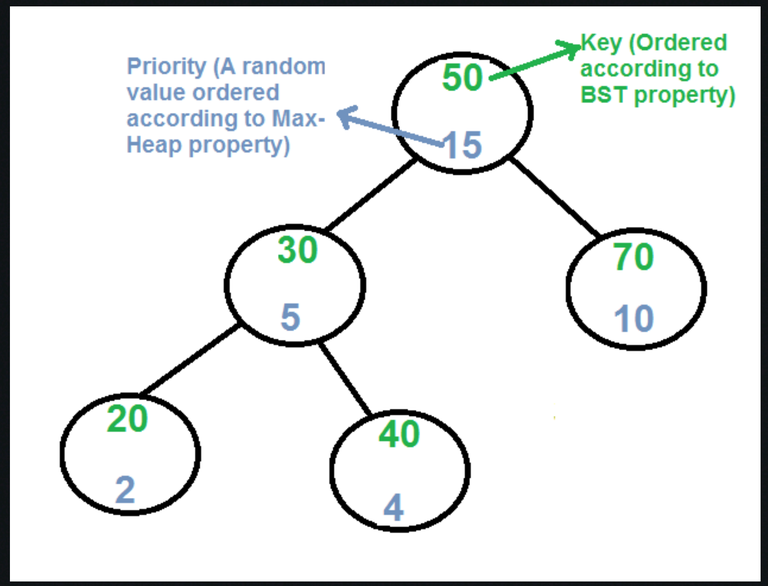
Market Place

Sean Chin, Nick Martin, Jacob Choi, CJ Balmaceda


depositphotos
Image ID: 169559440
www.depositphotos.com

Data Structures and Algorithms Used

Treap



Why use Treap?

- Combination of Heap and Binary Search Tree
  - Involves randomized priority value assignment
    - Results in a more balanced tree when inserting ordered values
  - Faster than Red-Black Trees

Treap

Program Snippet

Inventory prior to posting an item

```
Items:
Item: bike, Price: 10, Item Condition 4
Item: hat, Price: 15, Item Condition 1.5
Item: kite, Price: 5, Item Condition 3
Item: shoes, Price: 30, Item Condition 2.5
Item: watch, Price: 50, Item Condition 2
```

Inventory after posting an item

```
What is the name of the item?: chair
Enter your selling price: 10
Enter condition of the item 1-5(1 is poor condition, 5 is great condition): 5
Enter the category of the item:
"1" for Apparel
 "2" Vehicles
 "3" Accesory
 "4" Miscelaneous
4
Items:
Item: bike, Price: 10, Item Condition 4
Item: chair, Price: 10, Item Condition 5
Item: hat, Price: 15, Item Condition 1.5
Item: kite, Price: 5, Item Condition 3
Item: shoes, Price: 30, Item Condition 2.5
Item: watch, Price: 50, Item Condition 2

Thank you for posting the item!
```

Treap

Implementation - initialization

```
struct Item {
    std::string name;
    float price;
    float rating;
    std::string productType;

    Item(std::string name, float price, float rating,std::string productType) : name(name), price(price), rating(rating), productType(productType){}
};

template<typename T>
class TreapNode {
public:
    T data;
    int priority;
    TreapNode* left;
    TreapNode* right;

    TreapNode(T data) : data(data), priority(rand()), left(nullptr), right(nullptr) {}
};
```

```
template<typename T>
class Treap {
private:
    TreapNode<T>* root;
```

Treap

Implementation

```
void split(TreapNode<T>* current, std::string key, TreapNode<T>*& left, TreapNode<T>*& right) {
    if (current == nullptr) {
        left = nullptr;
        right = nullptr;
    } else if (key < current->data.name) {
        split(current->left, key, left, current->left);
        right = current;
    } else {
        split(current->right, key, current->right, right);
        left = current;
    }
}
```

```
TreapNode<T>* merge(TreapNode<T>* left, TreapNode<T>* right) {
    if (left == nullptr) return right;
    if (right == nullptr) return left;

    if (left->priority > right->priority) {
        left->right = merge(left->right, right);
        return left;
    } else {
        right->left = merge(left, right->left);
        return right;
    }
}
```

Treap

Implementation

```
void insert(TreapNode<T>*& current, TreapNode<T>* newNode) {
    if (current == nullptr) {
        current = newNode;
    } else if (newNode->priority > current->priority) {
        split(current, newNode->data.name, newNode->left, newNode->right);
        current = newNode;
    } else {
        insert(newNode->data.name < current->data.name ? current->left : current->right, newNode);
    }
}
```

Treap

Implementation

```
void remove(TreapNode<T>*& current, std::string key) {
    if (current == nullptr) return;

    if (current->data.name == key) {
        TreapNode<T>* temp = current;
        current = merge(current->left, current->right);
        delete temp;
    } else {
        remove(key < current->data.name ? current->left : current->right, key);
    }
}
```

Treap

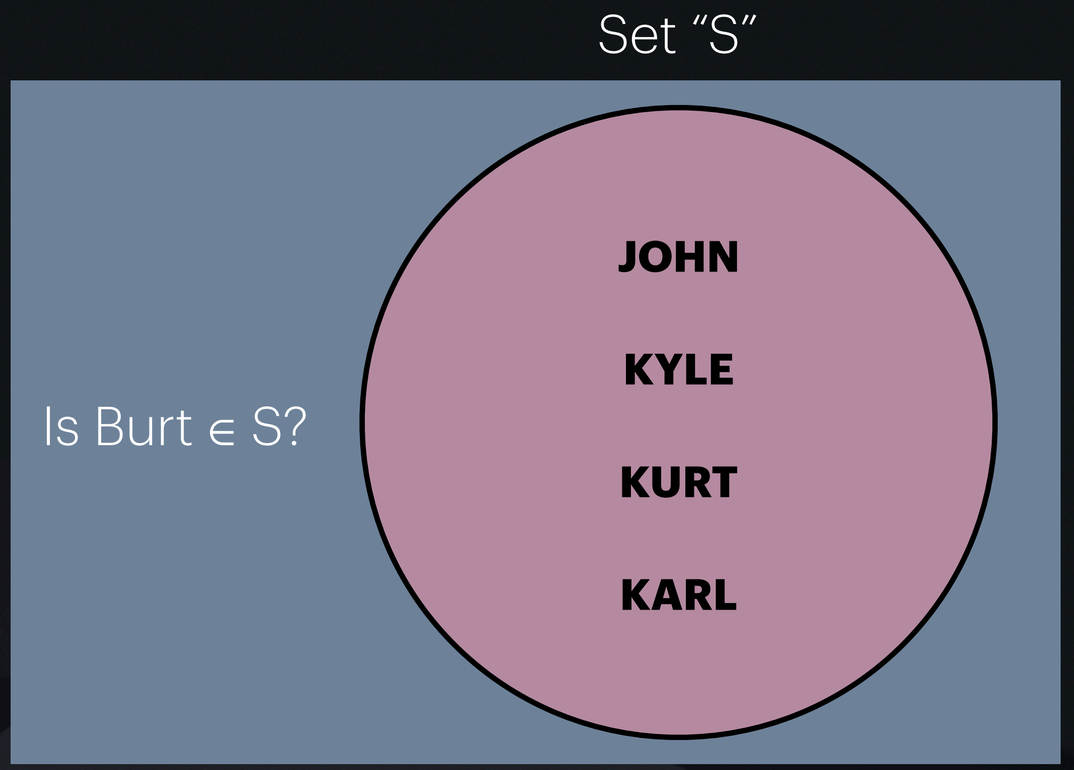
Implementation

```
std::vector<Item> inorderTraversal(TreapNode<T>* current, std::vector<Item>& items) {
    if (current != nullptr) {
        inorderTraversal(current->left, items);
        items.push_back(current->data);
        inorderTraversal(current->right, items);
    }
    return items;
}
```

Bloom Filter

Why use Bloom Filter?

- Primary use is for determining set membership
- Efficient space and time complexity
  - Time O(K)
    - K = number of hash functions
  - Space O(M)
    - M = number of bits in the array



Bloom Filter Implementation

Program snippet

```
Welcome to our marketplace simulation UI!
Do you have an account? (yes/no): yes
Please enter your username (or 'q' to quit): FrtNight
Username not found. Please try again.
Please enter your username (or 'q' to quit): fortnight
Please enter your password: password
```

Bloom Filter Implementation

Initialization

```
private:
    std::vector<bool> filter;
    int numHashes;
    int size;
    std::unordered_map<std::string, std::string> credentials;

public:
    BloomFilter(int numHashes, int size) : numHashes(numHashes), filter(size, false) {}
```

Bloom Filter Implementation

Credential verification

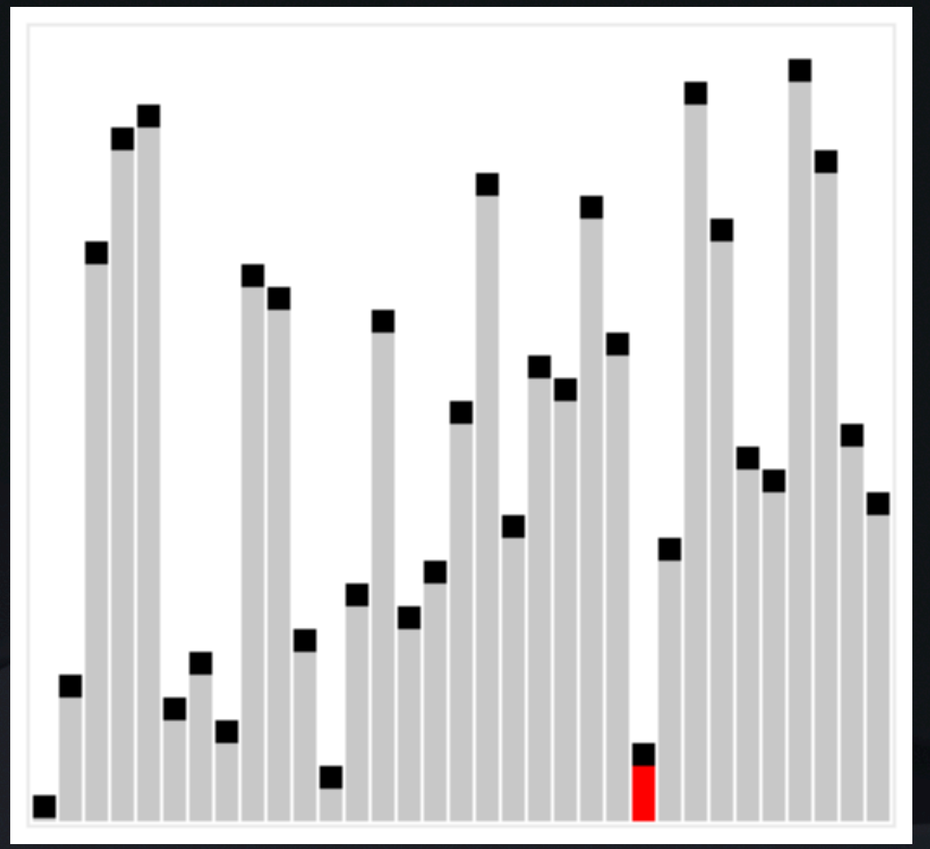
```
bool mightContain(const std::string& str) {
    for (int i = 0; i < numHashes; i++) {
        int hash1 = (hashFunction1(str) + i * hashFunction2(str)) % filter.size();
        if (!filter[hash1]) {
            return false;
        }
    }
    return true;
}
```

```
bool checkUsername(const std::string& username) {
    if (mightContain(username)) {
        return credentials.find(username) != credentials.end();
    }
    return false;
}
```

Shell Sort

Why use Shell Sort?

- Performs better than O(n^2)
    - O(n^1.5)
- Can be tuned to cater towards specific use cases



Shell Sort

Implementation

```
class ShellSort {
private:

    Treap<Item> treap;
    std::vector<Item> sortData;
    std::vector<Item> categoryData;

public:

   ShellSort(std::vector<Item>& unsortedData);
   ~ShellSort();
   void sortRatingAscend();
   void sortRatingAscend();
   bool category(std::string userCat);
   void sortPriceAscend();
   void sortPriceDescend();
   void display();
   void getData();

};

ShellSort::ShellSort(std::vector<Item>& unsortedData) : sortData(unsortedData) {}
```

Shell Sort

Program snippet

Unsorted

```
Item: bike, Price: 10, Item Condition 4
Item: hat, Price: 15, Item Condition 1.5
Item: kite, Price: 5, Item Condition 3
Item: shoes, Price: 30, Item Condition 2.5
Item: watch, Price: 50, Item Condition 2
```

Sorted

```
Would you like to buy or post an item for sale? (buy/post): buy
Great! What criteria would you like to sort the items by?
"1" for price
"2" for category
"3" for product condition
1
What order would you like that sorted in?
"1" for Low to High
"2" for High to Low
1
Items:
Item: kite, Price: 5, Item Condition 3
Item: bike, Price: 10, Item Condition 4
Item: hat, Price: 15, Item Condition 1.5
Item: shoes, Price: 30, Item Condition 2.5
Item: watch, Price: 50, Item Condition 2
```

Shell Sort

Implementation

```
void ShellSort::sortPriceAscend() {
    int n = sortData.size();
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            Item temp = sortData[i];
            int j;
            for (j = i; j >= gap && sortData[j - gap].price > temp.price; j -= gap) {
                sortData[j] = sortData[j - gap];
            }
            sortData[j] = temp;
        }
    }
}
```