VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA3 - HK242

Harmony BTL

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 4/2025

1 Harmony trong BTL1_HK242

Cho một class abstract - Trả lời các câu hỏi sau

```
template<class T>
class IList{
public:
    virtual ~IList(){};
    virtual void    add(T e)=0;
    virtual void    add(int index, T e)=0;
    virtual T       removeAt(int index)=0;
    virtual bool    removeItem(T item, void (*removeItemData)(T)=0)=0;
    virtual bool    empty()=0;
    virtual int     size()=0;
    virtual void    clear()=0;
    virtual T&      get(int index)=0;
    virtual int     indexOf(T item)=0;
    virtual bool    contains(T item)=0;
    virtual string  toString(string (*item2str)(T&)=0 )=0;
};
```

Class XArrayList thừa kế từ IList - Trả Lời 15 câu hỏi sau

```
template <class T>
class XArrayList : public IList<T> {
 public:
  class Iterator;  // forward declaration

 protected:
  T *data;
  int capacity;
  int count;
  bool (*itemEqual)(T &lhs, T &rhs);
  void (*deleteUserData)(XArrayList<T> *);


public:
 XArrayList(void (*deleteUserData)(XArrayList<T> *) = 0,
            bool (*itemEqual)(T &, T &) = 0, int capacity = 10);
 XArrayList(const XArrayList<T> &list);
 XArrayList<T> &operator=(const XArrayList<T> &list);
 ~XArrayList();

 // todo FUCNTION Inherit

 Iterator begin() { return Iterator(this, 0); }
 Iterator end() { return Iterator(this, count); }

protected:
 static bool equals(T &lhs, T &rhs, bool (*itemEqual)(T &, T &)) {
   if (itemEqual == 0)
     return lhs == rhs;
   else
     return itemEqual(lhs, rhs);
 }
 void checkIndex(int index);      // check validity of index for accessing
 void ensureCapacity(int index);  // auto-allocate if needed
 void copyFrom(const XArrayList<T> &list);
 void removeInternalData();
```

1. Điền code vào vị trí sau **/*Code 1*/**

```
template <class T>
void XArrayList<T>::add(T e) {
  this->ensureCapacity(this->count);
  this->data[/*Code 1*/] = e;
}
```

a) this.count b) this->count c) this->count++ d) ++this->count

2. Đầu là cách dùng foreach or đúng trong các câu sau khi duyệt qua tất cả phần tử

```
(1) for (auto it : list) {}
(2) for (auto it = list) {}
(3) for (auto it = list.begin(); it != list.end(); it++) {}
(4) for (auto it = list; it < list.end(); i--) {}
```

a) (1) b) (2) và (3) c) (1) và (2) d) Tất cả đều đúng

3. Điền code vào vị trí sau **/*Code 1*/**

```
template <class T>
T &XArrayList<T>::get(int index) {
  if (/*Code 1*/) {
    throw std::out_of_range("Index is out of range!");
  }

  return data[index];
}
```

a) index < 0 || index >= this->count b) index < 0 && index >= this->count
c) index <= 0 || index > this->count d) index <= 0 && index > this->count

4. Điền code vào vị trí sau **/*Code 1*/**

```
template <class T>
int XArrayList<T>::indexOf(T item) {
  for (int i = 0; i < this->count; ++i) {
    if (/*Code 1*/) {
      return i;
    }
  }

  return -1;
}
```

a) equals(data[i], item, itemEqual)
b) XArrayList::equals(data[i], item, itemEqual)
c) equals(data[i-1], item, itemEqual)
d) XArrayList::equals(data[i-1], item, itemEqual)

5. Điền code vào vị trí sau **/*Code 1*/ và /*Code 2*/ lần lượt là**

```
template <class T>
bool XArrayList<T>::removeItem(T item, void (*removeItemData)(T)) {
  for (int i = 0; i < this->count; i++) {
    if (XArrayList::equals(data[i], item, itemEqual)) {
      T temp = /*Code 1*/;
      if (removeItemData) {
        /*Code 2*/
      }
      return true;
    }
  }

  return false;
}
```

a) removeAt(i) và removeItemData(data[i]);
b) delete data[i] và removeItemData(data[i]);
c) removeAt(i) và removeItemData(data);
d) delete data[i] và removeItemData(data[i]);

6. Độ phức tạp của câu lệnh sau

```
for (int i = 0; i < xarray.size(); i++) cout << xarray.get(i);
```

a) N
b) N^2
c) N^3
d) 2^N

7. Điền vào chỗ trống để tìm tất cả sinh viên có tuổi bằng `targetAge`:

```
struct Student {string name; int age;};
XArrayList<Student> findByAge(XArrayList<Student> &list, int
 ↪  targetAge) {
    XArrayList<Student> result;
    for (int i = 0; i < list.size(); ++i) {
        if (_______) {
            result.add(list.get(i));
        }
    }
    return result;
}
```

a) `list.get(i).age == targetAge`
b) `(*list.get(i)).age == targetAge`
c) `list[i].age == targetAge`
d) `list.get(i)->age == targetAge`

8. Điền vào chỗ trống để xóa sinh viên có tuổi bằng `targetAge` khỏi danh sách:

```
struct Student {string name; int age;};
bool cmp(Student &a, Student &b) {
    return a.age == b.age;
```

```
}

void removeByAge(XArrayList<Student> &list, int targetAge) {
    Student temp = {"", targetAge};
    list._________;
}
```

a) `removeItem(temp)`
b) `removeItem(temp, cmp)`
c) `removeItem(temp, cmp)`
d) `removeItem(temp, nullptr)`

9. Diền vào chỗ trống để xóa tất cả sinh viên có tên là "An" khỏi danh sách:

```
struct Student {string name; int age;};
void removeByName(IList<Student> &list) {
    for (int i = 0; i < list.size(); ) {
        if (list.get(i).name == "An") {
            ____________;
        } else {
            ++i;
        }
    }
}
```

a) `list.removeItem(list.get(i))`
b) `list.removeAt(i)`
c) `list.removeItem("An")`
d) `delete list.get(i)`

10. Diền vào chỗ trống để kiểm tra xem danh sách có chứa phần tử có tên là "An" hay không:

```
struct Student {string name; int age;};
bool hasStudentNamedAn(IList<Student> &list) {
    for (int i = 0; i < _______; ++i) {
        if (list.get(i).name == "An") return true;
    }
    return false;
}
```

a) `list.size()`
b) `list.indexOf()`
c) `list.contains()`
d) `list.get()`

11. Điền vào chỗ trống để in ra tất cả phần tử của ma trận:

```
void printMatrix(XArrayList<XArrayList<int>> &matrix) {
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = 0; j < matrix.get(i).size(); ++j) {
            cout << ___________ << " ";
        }
        cout << endl;
    }
}
```

a) `matrix.get(i).get(j)`
b) `matrix[i][j]`
c) `matrix.get(j).get(i)`
d) `matrix.get(i, j)`

12. Điền vào chỗ trống để tính tổng các phần tử trên đường chéo chính của ma trận vuông:

```
int sumDiagonal(XArrayList<XArrayList<int>> &matrix) {
    int sum = 0;
    for (int i = 0; i < matrix.size(); ++i) {
        sum += ____________;
    }
    return sum;
}
```

a) `matrix.get(i).get(i)`
b) `matrix.get(i, i)`
c) `matrix[i][i]`
d) `matrix.get(i).get(0)`

13. Độ phức tạp thời gian của đoạn mã sau là bao nhiêu?

```
int total = 0;
for (int i = 0; i < matrix.size(); ++i) {
    for (int j = 0; j < matrix.get(i).size(); ++j) {
        total += matrix.get(i).get(j);
    }
}
```

a) O(N)
b) $O(N^2)$
c) $O(N \cdot M)$
d) $O(\log N)$

14. Điền vào chỗ trống để sắp xếp các phần tử trong ma trận theo thứ tự tăng dần (sử dụng chỉ hai vòng 'for'):

```
void sortMatrix(IList<XArrayList<int>> &matrix) {
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix.get(i).size(); ++j) {
            if (__________) {
                int temp = matrix.get(i).get(j);
                matrix.get(i).get(j) = matrix.get(i).get(i);
                matrix.get(i).get(i) = temp;
```

```
            }
        }
    }
}
```

a) `matrix.get(i).get(j) > matrix.get(i).get(i)`
b) `matrix.get(i).get(j) < matrix.get(i).get(i)`
c) `matrix.indexOf(matrix.get(i).get(j)) > matrix.indexOf(matrix.get(i).get(i))`
d) `matrix.contains(matrix.get(i).get(j)) < matrix.contains(matrix.get(i).get(i))`

15. Diền vào chỗ trống để sao chép ma trận 2 chiều (kiểu 'IList<IList<int»') từ một đối tượng khác:

```
IList<IList<int>> &operator=(const IList<IList<int>> &matrix) {
    if (this == &matrix) return *this;
    this->clear();  // Xóa dữ liệu cũ
    for (int i = 0; i < matrix.size(); ++i) {
        IList<int> newRow;
        for (int j = 0; j < matrix.get(i).size(); ++j) {
            newRow.add(matrix.get(i).get(j));
        }
        ------------;
    }
    return *this;
}
```

a) `this->add(newRow)`
b) `matrix.add(newRow)`
c) `this->get(i).add(newRow)`
d) `matrix.get(i).add(newRow)`

Class DLinkedList thừa kế từ IList (private trong khóa cuối kì) - Trả Lời 10 câu hỏi sau

```
template <class T>
class DLinkedList : public IList<T> {
 protected:
  Node *head;  // this node does not contain user's data
  Node *tail;  // this node does not contain user's data
  int count;
  bool (*itemEqual)(T &lhs, T &rhs);
  void (*deleteUserData)(DLinkedList<T> *);

 public:
  DLinkedList(void (*deleteUserData)(DLinkedList<T> *) = 0,
              bool (*itemEqual)(T &, T &) = 0);
  DLinkedList(const DLinkedList<T> &list);
  DLinkedList<T> &operator=(const DLinkedList<T> &list);
  ~DLinkedList();

  // TODO Inherit IList

  Iterator begin() { return Iterator(this, true); }
  Iterator end() { return Iterator(this, false); }
  BWDIterator bbegin() { return BWDIterator(this, true); }
  BWDIterator bend() { return BWDIterator(this, false); }

 protected:
  static bool equals(T &lhs, T &rhs, bool (*itemEqual)(T &, T &));
  void copyFrom(const DLinkedList<T> &list);
  void removeInternalData();
  Node *getPreviousNodeOf(int index);
```

```
public:
 class Node {
  public:
   T data;
   Node *next;
   Node *prev;
   friend class DLinkedList<T>;
  public:
   Node(Node *next = 0, Node *prev = 0);
   Node(T data, Node *next = 0, Node *prev = 0);
 };
 class Iterator {
  private:
   DLinkedList<T> *pList;
   Node *pNode;

  public:
   Iterator(DLinkedList<T> *pList = nullptr, bool begin = true);
   Iterator &operator=(const Iterator &iterator);
   bool operator!=(const Iterator &iterator) const;
   Iterator &operator++();
 };

 class BWDIterator {
  private:



        DLinkedList<T> *pList;
        Node *pNode;

       public:
        BWDIterator(DLinkedList<T> *pList = nullptr, bool begin = true);
        BWDIterator &operator=(const BWDIterator &iterator);
        T &operator*();
        bool operator!=(const BWDIterator &iterator) const;
        BWDIterator &operator++();
      };
    };
```

16. Điền code vào vị trí sau **/*Code 1*/**

```
template <class T>
void DLinkedList<T>::add(T e) {
  Node *newNode = new Node(e);

  if (this->head == nullptr) {
    /* Code 1 */
  } else {
    newNode->prev = this->tail;
    /* Code 2 */
    this->tail = newNode;
  }
  this->count++;
}

template <class T>
void DLinkedList<T>::add(int index, T e) {
  if (index < 0 || index > this->count) {
    // TODO
  }
  Node *newNode = new Node(e);
  if (this->head == nullptr) {
    // TODO
  } else if (index == 0) {
    // TODO
  } else if (index == this->count) {
    // TODO
  } else {
    Node *current = this->head;
    for (int i = 0; i < index; i++) {
      current = current->next;
    }
    newNode->next = current;
    newNode->prev = /*code 3*/;
    current->prev->next = newNode;
    /*code 4*/ = newNode;
  }
  this->count++;
}
```

a) this->head = newNode;
b) this->tail = newNode;
c) this->head = this->tail = null;
d) this->head = this->tail = newNode;

17. Điền code vào vị trí sau **/*Code 2*/** trong câu 1

a) this->tail->next = newNode;
b) this->tail->prev = newNode;
c) this->head->next = newNode;
d) this->head->next = newNode;

18. Điền code vào vị trí sau **/*Code 3*/** trong câu 1

a) current->prev
b) current->next
c) newNode->prev
d) newNode->next

19. Điền code vào vị trí sau **/*Code 4 */** trong câu 1

a) current->prev
b) current->next
c) newNode->prev
d) newNode->next

20. Diền code vào vị trí sau **/*Code 1*/**

```
template <class T>
T DLinkedList<T>::removeAt(int index) {
  if (index < 0 || index >= this->count) {
    throw out_of_range("Index out of range!");
  }

  Node *temp = getPreviousNodeOf(index);

  if (temp == nullptr) {
    temp = this->head;
    /*Code 1*/
    if (this->head != nullptr) {
      this->head->prev = nullptr;
    } else {
      /*Code 2*/
    }
  } else {
    // TODO
  }

  T data = temp->data;
  delete temp;
  this->count--;

  return data;
}
```

a) this->head = this->head->next;
b) this->tail = this->head->next;
c) this->head = this->head->prev;
d) this->tail = this->head->prev;

21. Diền code vào vị trí sau **/*Code 2*/** trong câu 5

a) this->tail = nullptr;
b) this->head = nullptr;
c) this->tail = this->head->next;
d) this->head = this->tail;

22. Độ phức tạp tối ưu nhất của **empty** và **size**

a) O(1)
b) O(N*N)
c) O(log(N))
d) O(N)

23. Kết quả đoạn code sau

```
DLinkedList<int> list;
list.add(1);
list.add(2);
list.add(3);

cout << "(";
for (auto it = list.bbegin(); it != list.bend(); ++it) {
    if (it != list.bbegin()) output << ", ";
    cout << *it;
}
cout << ")";
```

a) (1,2,3)
b) (1,2)
c) (2,3)
d) (3,2,1)

24. Kết quả đoạn code sau

```
DLinkedList<int> list;
list.add(1);
auto it = list.bbegin();
list.add(0, 2);
list.add(3);
it++;
cout << *it;
```

a) 2 b) 3 c) 1 d) Lỗi biên dịch

25. Kết quả đoạn code sau

```
DLinkedList<int> list;
list.add(1);
auto it = list.end();
list.add(0, 2);
list.add(3);
it++;
cout << *it;
```

a) 2 b) 3 c) 1 d) Lỗi biên dịch

2 Harmony trong BTL2_HK242

10 câu hỏi về hash

```
template <class K, class V>
class xMap : public IMap<K, V> {
 public:
  class Entry;  // forward declaration

 protected:
  DLinkedList<Entry*>* table;  // array of DLinkedList objects
  int capacity;                // size of table
  int count;                   // number of entries stored hash-map
  float loadFactor;            // define max number of entries can be stored (<
                               // (loadFactor * capacity))

  int (*hashCode)(K&, int);  // hasCode(K key, int tableSize): tableSize means capacity
  //TODO
 public:
  // Entry: BEGIN
  class Entry {
   private:
    K key;
    V value;
    friend class xMap<K, V>;

   public:
    Entry(K key, V value) {
      this->key = key;
      this->value = value;
    }
  };
  // Entry: END

 public:
    //TODO
}
```

Nếu count > loadFactor * capacity thì sẽ cập nhật capacity

1. Điền code vào vị trí sau **/*Code 1*/** Hàm lấy ra danh sách các key

```
template <class K, class V>
DLinkedList<K> xMap<K, V>::keys() {
  DLinkedList<K> keyList;

  for (int index = 0; index < capacity; index++) {
    DLinkedList<Entry*> entrys = /*Code 1*/;
    for (auto entry = entrys.begin(); entry != entrys.end();
     ↪  entryNode++) {
      keyList.add(/*Code 2*/);
    }
  }

  return keyList;
}
```

a) table[index] b) table[index - 1] c) table[index + 1] d) table[index–]

2. Điền code vào vị trí sau **/*Code 2*/** ở câu trên

a) (*entry)->key b) (*entry).key c) (*entry)->value d) (*entry).value

3. Giả sử hàm hash code được truyền vào xMap như sau, load factor luôn đảm bảo và capacity=20

```
int intKeyHash(int& key, int capacity) { return key % (capacity -
 ↪  1); }
```

với mảng sau được truyền vào [1,31,21,2,8,9,41,61,51] thì vị trí nào sẽ được nhiều giá trị nhất

a) index 0 b) index 2 c) index 1 d) index 3

4. Giả sử bạn đang xây dựng một hệ thống quản lý thông tin người dùng cho một ứng dụng. Mỗi người dùng có một ID duy nhất và các thuộc tính như tên, email, và số điện thoại. Bạn muốn sử dụng một bảng băm (hash map) để lưu trữ và tìm kiếm thông tin người dùng theo ID.

Giả sử bạn sử dụng 'xMap' để lưu trữ các đối tượng 'User' với khóa là ID người dùng (kiểu 'int') và giá trị là đối tượng 'User' (kiểu 'User'), trong đó 'User' có các thuộc tính 'name', 'email' và 'phone'.

Đoạn mã dưới đây mô tả cách sử dụng 'xMap' để lưu trữ và tìm kiếm thông tin người dùng:

```cpp
struct User {
    std::string name;
    std::string email;
    std::string phone;
};

xMap<int, User> userMap;

// Hàm thêm người dùng vào bảng băm
void addUser(int id, const std::string& name, const std::string&
 ↪  email, const std::string& phone) {
    User user = {name, email, phone};
    userMap.put(id, user);
}

// Hàm tìm kiếm thông tin người dùng theo ID
User getUserById(int id) {
    return userMap.get(id);
}
```

Trong ứng dụng này, bảng băm sẽ giúp bạn nhanh chóng tìm kiếm thông tin người dùng bằng cách sử dụng ID. Câu hỏi sau đây yêu cầu bạn xác định lý do tại sao sử dụng 'xMap' lại hiệu quả trong tình huống này.

a) 'xMap' cho phép tìm kiếm thông tin người dùng theo ID nhanh chóng với độ phức tạp O(1) trung bình.

b) 'xMap' yêu cầu phải duyệt qua toàn bộ danh sách người dùng để tìm kiếm một ID.

c) 'xMap' không hỗ trợ lưu trữ các đối tượng phức tạp như 'User'.

d) 'xMap' chỉ có thể sử dụng với kiểu dữ liệu nguyên thủy như số nguyên.

5. Giả sử bạn đang xây dựng một hệ thống quản lý sản phẩm trong một cửa hàng trực tuyến. Mỗi sản phẩm có một mã sản phẩm duy nhất và các thông tin như tên, giá, và số lượng tồn kho. Bạn quyết định sử dụng 'xMap' để quản lý các sản phẩm trong cửa hàng, với khóa là mã sản phẩm (kiểu 'string') và giá trị là đối tượng 'Product' (kiểu 'Product'), trong đó 'Product' chứa các thuộc tính 'name', 'price', và 'stock'.

   Đoạn mã dưới đây mô tả cách sử dụng 'xMap' để lưu trữ, tìm kiếm và xóa thông tin sản phẩm trong cửa hàng:

```
struct Product {
    std::string name;
    double price;
    int stock;
};

xMap<std::string, Product> productMap;

// Hàm thêm sản phẩm vào bảng băm
void addProduct(const std::string& code, const std::string& name,
 ↪  double price, int stock) {
    Product product = {name, price, stock};
    productMap.put(code, product);
}

// Hàm tìm kiếm sản phẩm theo mã
Product getProductByCode(const std::string& code) {
    return productMap.get(code);
}

// Hàm xóa sản phẩm theo mã
bool removeProduct(const std::string& code) {
    return productMap.remove(code);
}

// Hàm kiểm tra xem mã sản phẩm có tồn tại trong bảng băm không
bool isProductAvailable(const std::string& code) {
    return productMap.containsKey(code);
}

// Hàm lấy thông tin tất cả sản phẩm dưới dạng chuỗi
std::string getAllProducts() {
    return productMap.toString();
}
```

   Giả sử bạn đã thêm một số sản phẩm vào hệ thống. Câu hỏi sau yêu cầu bạn xác định hành động nào sau đây sẽ là đúng khi sử dụng các hàm của 'xMap'.

a) Dùng 'put' để thêm sản phẩm mới vào bảng băm và 'get' để lấy thông tin sản phẩm khi cần.
b) Dùng 'remove' để xóa sản phẩm khỏi bảng băm và 'containsKey' để kiểm tra nếu sản phẩm có tồn tại trong hệ thống.
c) Dùng 'get' để lấy thông tin sản phẩm nhưng không kiểm tra xem sản phẩm có tồn tại trước không.
d) Cả ba phương án trên đều đúng trong tất cả các tình huống.

6. Điền vào chỗ trống để trả về tần suất xuất hiện của các ký tự trong chuỗi bằng cách sử dụng 'xMap':

```
xMap<char, int> getCharFrequency(const string &str) {
    xMap<char, int> frequencyMap;
    for (char c : str) {
        if (frequencyMap.containsKey(c)) {  // Kiểm tra nếu ký tự đã có trong bản đồ
            ____________;  // Cập nhật tần suất ký tự
        } else {
            frequencyMap.put(c, 1);  // Thêm ký tự vào bản đồ với tần suất là 1
        }
    }
    return frequencyMap;
}
```

a) `frequencyMap.put(c, frequencyMap.get(c) + 1);`
b) `frequencyMap.put(c, frequencyMap.get(c) - 1);`
c) `frequencyMap.add(c, frequencyMap.get(c) + 1);`
d) `frequencyMap.put(c, 1);`

7. Giả sử bạn quản lý kho hàng của một cửa hàng và muốn theo dõi số lượng các sản phẩm trong kho. Sử dụng 'xMap' để lưu trữ tên sản phẩm (key) và số lượng (value). Điền vào chỗ trống để cập nhật số lượng sản phẩm trong kho:

```
void updateInventory(xMap<string, int> &inventory, const string &product, int quantity) {
    if (inventory.containsKey(product)) {
        // Cập nhật số lượng sản phẩm
        ___________
    } else {
        // Thêm sản phẩm mới vào kho
        inventory.put(product, quantity);
    }
}
```

a) `inventory.put(product, inventory.get(product) + quantity);`
b) `inventory.add(product, inventory.get(product) + quantity);`
c) `inventory.put(product, 0);`
d) `inventory.put(product, quantity);`

8. Giả sử bạn đang quản lý kho hàng của một cửa hàng, trong đó 'xMap' lưu trữ tên sản phẩm (key) và số lượng sản phẩm (value). Điền vào chỗ trống để tìm sản phẩm có số lượng lớn nhất trong kho:

```
string getMaxProduct(xMap<string, int> &inventory) {
    string maxProduct;
    int maxQuantity = 0;
    for (auto &entry : inventory) {
        if (entry.value > maxQuantity) {
            ______________
            maxProduct = entry.key;
        }
    }
    return maxProduct;
}
```

a) `maxQuantity = entry.value;`
b) `maxQuantity = entry.key;`
c) `maxQuantity = inventory.get(entry.key);`
d) `maxProduct = entry.value;`

9. Giả sử bạn đang quản lý hệ thống bán hàng của một cửa hàng trực tuyến, trong đó 'xMap' lưu trữ tên sản phẩm (key) và số lượng sản phẩm (value). Mỗi khi có đơn hàng, bạn cần giảm số lượng sản phẩm trong kho theo số lượng sản phẩm được mua. Điền vào chỗ trống để cập nhật số lượng sản phẩm trong kho sau khi nhận được đơn hàng:

```
void updateInventory(xMap<string, int> &inventory, string product,
 ↪  int quantity) {
    if (inventory.containsKey(product)) {
        int currentQuantity = inventory.get(product);
        if (currentQuantity >= quantity) {
            currentQuantity -= quantity;
            ___________  // Cập nhật số lượng
        } else {
            cout << "Không đủ số lượng sản phẩm: " << product;
        }
    } else {
        cout << "Sản phẩm không tồn tại trong kho: " << product;
    }
}
```

a) `currentQuantity -= quantity;`
b) `currentQuantity = quantity;`
c) `inventory.put(product, quantity);`
d) `inventory.put(product, currentQuantity);`

10. Giả sử bạn đang quản lý hệ thống điểm thưởng cho khách hàng trong một cửa hàng, nơi 'xMap' lưu trữ ID khách hàng (key) và số điểm thưởng của họ (value). Mỗi khi một giao dịch được thực hiện, bạn cần cập nhật điểm thưởng của khách hàng theo số điểm mới. Điền vào chỗ trống để xử lý việc cập nhật điểm thưởng:

Võ Tiến (DSA3 - HK242 GK & CK)

```
void updateRewardPoints(xMap<int, int> &rewards, int customerId,
    int points) {
    if (rewards.containsKey(customerId)) {
        int currentPoints = rewards.get(customerId);
        currentPoints += points;  // Thêm điểm thưởng mới
        // Cập nhật điểm thưởng mới
        rewards.put(customerId, currentPoints);
    } else {
        -------- // Nếu khách hàng chưa có điểm thưởng, thêm mới
    }
}
```

a) `rewards.put(customerId, currentPoints);`
b) `rewards.put(customerId, points);`
c) `rewards.get(customerId) += points;`
d) `rewards.put(customerId, rewards.get(customerId) + points);`

8 câu hỏi về heap

```
template <class T>
class Heap : public IHeap<T> {
 public:
  class Iterator;  // forward declaration

 protected:
  T *elements;   // a dynamic array to contain user's data
  int capacity;  // size of the dynamic array
  int count;     // current count of elements stored in this heap
  int (*comparator)(T &lhs, T &rhs);           // see above
  void (*deleteUserData)(Heap<T> *pHeap) = 0;  // see above

 public:
    //TODO
};
```

1. Điền code vào vị trí sau **/*Code 1*/** Hàm thêm 1 phần tử vào heap

```
template <class T>
void Heap<T>::push(T item) {
  ensureCapacity(count + 1);

  elements[/*Code 1*/] = item;
  reheapUp(count - 1);
}
```

a) ++count b) count- - c) count++ d) count- -

2. Điền code vào vị trí sau **/*Code 1*/** Hàm thêm xóa phần tử có giá trị giống giá trị truyền vào trong heap

```
template <class T>
void Heap<T>::remove(T item, void (*removeItemData)(T)) {
  int foundIdx = getItem(item);

  // TODO not find

  elements[foundIdx] = elements[/*Code 1*/];
  /*Code 2*/
  // TODO
}
```

a) ++count b) count- - c) count++ d) count- -

3. Điền code vào vị trí sau **/*Code 2*/** ở câu trên

a) reheapDown(foundIdx); b) reheapDown(foundIdx - 1);
c) reheapDown(foundIdx + 1); d) count- -

4. Điền code vào vị trí sau **/*Code 1*/** Hàm reheap down trong heap

```
template <class T>
void Heap<T>::reheapDown(int position) {
  int leftChild = /*Code 1*/;
```



```
  int rightChild = position * 2 + 2;
  int target = position;

  // TODO

  if (rightChild < count &&
      compare(elements[target], elements[rightChild]) > 0) {
    target = rightChild;
  }

  if (target != position) {
    swap(position, target);
    /*Code 2*/
  }
}
```

a) position * 2 + 3 b) position * 2 + 1 c) position * 2 + 2 d) position * 2

5. Điền code vào vị trí sau **/*Code 2*/** ở câu trên

   a) reheapDown(target);
   b) reheapDown(leftChild);
   c) reheapDown(rightChild);
   d) count- -

6. Độ phức tạp trung bình của hàm `push(item)` trong heap là?

   a) $O(1)$ b) $O(n)$ c) $O(n \log n)$ d) $O(\log n)$

7. Độ phức tạp tệ nhất của hàm `remove(item)` trong heap là?

   a) $O(\log n)$ b) $O(n)$ c) $O(n + \log n)$ d) $O(1)$

8. Điền code vào vị trí sau **/*Code 1*/** trong hàm `heapSort`.
   Gợi ý: Hàm `heapify(arr, n, i)` đảm bảo rằng cây nhị phân có gốc tại chỉ số `i` là một heap hợp lệ, với `n` là kích thước vùng đang xét. Hàm này thường được gọi lại sau khi phần tử lớn nhất (ở gốc) bị đẩy xuống cuối mảng.

```
template <class T>
void heapSort(T arr[], int n) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract elements
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        /*Code 1*/
    }
}
```

   a) heapify(arr, i, 0);
   b) heapify(arr, 0, i);
   c) heapify(arr, n, i);
   d) heapify(arr, 1, i);

8 câu hỏi về Tree

```
template<int treeOrder>
class HuffmanTree {
public:
    struct HuffmanNode {
        char symbol;
        int freq;
        XArrayList<HuffmanNode*> children;
        int id;
        HuffmanNode(char s, int f) : symbol(s), freq(f), id(-1) {} //Leaf node
        HuffmanNode(int f, const XArrayList<HuffmanNode*>& childs)
            : symbol('\0'), freq(f), id(-1), children(childs) {}; //Internal node
        void setId(int nodeId) { id = nodeId; }
    };

    HuffmanTree();
    ~HuffmanTree();

    void build(XArrayList<pair<char, int>>& symbolsFreqs);
    void generateCodes(xMap<char, std::string>& table);
    std::string decode(const std::string& huffmanCode);
}
```

1. Cho danh sách ký tự và tần số sau:

   `{'A', 3}, {'B', 1}, {'C', 2}, {'D', 1}`

   Với `treeOrder = 3`, đếm số node lá.

   Gợi ý: Dùng cây Huffman bậc 3 (mỗi nút có tối đa 3 con). Nếu cần, thêm các nút giả (dummy) có tần số 0 để đảm bảo tổng số nút lá thỏa mãn công thức $(L-1) \bmod (k-1) = 0$*, trong đó* L *là số nút lá (kể cả giả), và* $k =$ `treeOrder`.

   a) 5 Node b) 4 Node c) 6 Node d) 2 Node

2. Cho danh sách ký tự và tần số sau:

   `{'A', 3}, {'B', 1}, {'C', 2}, {'D', 1}`

   Với `treeOrder = 3`, Kết quả câu khi `build`.

   a) `7(2(D, B, Dummy), C, A)`
   b) `7(4(A, B, Dummy), C, D)`
   c) `7(4(D, A, Dummy), C, B)`
   d) `7(3(D, C, Dummy), B, A)`

3. Cho danh sách ký tự và tần số sau:

   `{'A', 3}, {'B', 1}, {'C', 2}, {'D', 1}`

   Với `treeOrder = 2`, Chiều cao của cây (node đầu tiên có h=1).

   a) 4 b) 3 c) 5 d) 2

4. Cho danh sách ký tự và tần số sau:

   `{'A', 3}, {'B', 1}, {'C', 2}, {'D', 1}`

   Với `treeOrder = 2`, Liệt kê các node nội (không tính node root).

   a) `[2], [4]` b) `[3], [4]` c) `[5], [4]` d) `[5], [2]`

```
void build(XArrayList<pair<char, int>>& symbolsFreqs)
{
    // TODO xử lí node giả

    Heap<HuffmanNode*> heap(compareNode);

    int nodeIdCounter = 0;

    // Khởi tạo các nút lá
    for (int i = 0; i < symbolsFreqs.size(); ++i) {
        char ch = symbolsFreqs.get(i).first;
        int freq = symbolsFreqs.get(i).second;
        HuffmanNode* node = new HuffmanNode(ch, freq);
        node->setId(nodeIdCounter++);
        /* CODE 1 */
    }

    // Lặp lại đến khi heap chỉ còn một node duy nhất (node gốc)
    while (heap.size() > 1) {
        XArrayList<HuffmanNode*> nodes;

        int takeCount = std::min(treeOrder, heap.size());
        for (int i = 0; i < takeCount; ++i) {
            nodes.add(heap.pop());
        }

        int sumFreq = 0;
        for (int i = 0; i < nodes.size(); ++i) {
            sumFreq += nodes.get(i)->freq;
        }

        HuffmanNode* parent = /* CODE 2 */;
        parent->setId(nodeIdCounter++);
        heap.push(parent);
    }
    root = heap.pop();
}
```

5. Điền code vào vị trí sau **/*Code 1*/** ở câu trên

    a) `heap.push(node);`
    b) `heap.pop(node);`
    c) `heap.get(node);`
    d) `heap.push(freq);`

6. Điền code vào vị trí sau **/*Code 2*/** ở câu trên

    a) `new HuffmanNode(sumFreq, nodes);`
    b) `new HuffmanNode(0, nodes);`
    c) `new HuffmanNode(sumFreq + 1, nodes);`
    d) `new HuffmanNode(1, nodes);`

```
void dfsGenerateCode(HuffmanNode* node, const std::string& code,
    xMap<char, std::string>& table) {
    if (!node) return;

    if (node->children.size() == 0) {
        table.put(node->symbol, code);
        return;
    }

    for (int i = 0; /*Code 1*/; ++i) {
        char digit = (i < 10) ? ('0' + i) : ('a' + (i - 10));
        dfsGenerateCode(
            node->children.get(i),
            /*Code 2*/,
            table
        );
    }
}


void generateCodes(xMap<char, std::string> &table)
{
    dfsGenerateCode(root, "", table);
}
```

7. Điền code vào vị trí sau **/*Code 1*/** ở câu trên

a) `i < node->children.size()`
b) `i < node.children.size() - 1`
c) `i < node->children.size()`
d) `i < node.children.size() - 1`

8. Điền code vào vị trí sau **/*Code 2*/** ở câu trên

a) `code + digit`
b) `digit + code`
c) `code`
d) `digit`