BÀI TẬP 3

CHUYÊN ĐỀ TỔ CHỨC DỮ LIỆUKÌ 2 2022-2023, HỆ ĐÀO TẠO TỪ XA

MSSV: 21880159

Họ và Tên: Nguyễn Hữu Vinh

**1.**

a) Cấu trúc dữ liệu hàng đợi ưu tiên sử dụng heap:

|  |
| --- |
| #include <iostream>  using namespace std;  struct PriorityQueue {  private:  int\* heap;  int\* priority;  int size;  int capacity;  public:  //constructor  PriorityQueue(int cap = 1000) {  capacity = cap;  size = 0;  heap = new int[capacity];  priority = new int[capacity];  }  int parent(int i) { return (i - 1) / 2; }  int leftChild(int i) { return 2 \* i + 1; }  int rightChild(int i) { return 2 \* i + 2; }  void swap(int& a, int& b) {  int temp = a;  a = b;  b = temp;  }  //insert  void enqueue(int value, int p) {  if (size < capacity) {  int i = size++;  heap[i] = value;  priority[i] = p;  while (i != 0 && priority[parent(i)] > priority[i]) {  swap(heap[i], heap[parent(i)]);  swap(priority[i], priority[parent(i)]);  i = parent(i);  }  }  else {  cout << "Priority queue is full!\n";  }  }  //remove  int dequeue() {  if (size == 0) {  cout << "Priority queue is empty!\n";  return -1;  }  else {  int root = heap[0];  heap[0] = heap[--size];  priority[0] = priority[size];  minHeapify(0);  return root;  }  }  void minHeapify(int i) {  int smallest = i;  int l = leftChild(i);  int r = rightChild(i);  if (l < size && priority[l] < priority[smallest]) {  smallest = l;  }  if (r < size && priority[r] < priority[smallest]) {  smallest = r;  }  if (smallest != i) {  swap(heap[i], heap[smallest]);  swap(priority[i], priority[smallest]);  minHeapify(smallest);  }  }  //remove max  int removeMax() {  if (size == 0) {  cout << "Priority queue is empty!\n";  return -1;  }  else {  int maxIndex = 0;  for (int i = 1; i < size; i++) {  if (priority[i] > priority[maxIndex]) {  maxIndex = i;  }  }  int maxValue = heap[maxIndex];  heap[maxIndex] = heap[--size];  priority[maxIndex] = priority[size];  minHeapify(maxIndex);  return maxValue;  }  }  //print priority queue  void display() {  if (size == 0) {  cout << "Priority queue is empty!\n";  return;  }  cout << "Priority queue: ";  for (int i = 0; i < size; i++) {  cout << "(" << heap[i] << "," << priority[i] << ")" << " ";  }  cout << endl;  }  }; |

b) Để sắp mảng tăng dần ta chỉ cần viết thêm hàm sortArray(), vì đây là min heap nên ta chỉ cần tạo một hàng đợi ưu tiên và đưa tất cả phần tử của mảng vào hàng đợi ưu tiên đó enqueue(). Sau đó, ta lấy tất cả các phần tử ra khỏi hàng đợi ưu tiên lúc đầu và gán vào từng phần tử của mảng dequeue() ta sẽ có được mảng sắp xếp tăng dần.

|  |
| --- |
| //sort array  void sortArray(int arr[], int n) {  PriorityQueue pq(n);  for (int i = 0; i < n; i++) {  pq.enqueue(arr[i], arr[i]);  }  for (int i = 0; i < n; i++) {  arr[i] = pq.dequeue();  }  }  //print array  void displayArray(int arr[], int n) {  cout << "Sorted array: ";  for (int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  } |

Chạy thử chương trình:

|  |
| --- |
| int main() {  cout << "Chay thu cau truc hang doi uu tien: " << endl;  PriorityQueue pq = PriorityQueue();  pq.dequeue();  pq.enqueue(4, 2);  pq.enqueue(1, 4);  pq.enqueue(3, 1);  pq.enqueue(9, 5);  pq.enqueue(7, 3);  pq.display();  cout << "Value has max priority: " << pq.removeMax();  cout << endl;  cout << "-----------------------" << endl;  cout << "Sap xep mang tang dan: " << endl;  int arr[] = { 4, 1, 3, 9, 7 };  int n = sizeof(arr) / sizeof(arr[0]);  cout << "Original array: ";  for (int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  sortArray(arr, n);  displayArray(arr, n);  return 0;  } |

**Kết quả chạy thử:**

Text

Description automatically generated

**2.**

**Cấu trúc dữ liệu túi sử dụng mảng:**

|  |
| --- |
| #include <iostream>  using namespace std;  struct Data {  int value;  int count;  };  struct Bag {  private:  Data data[1000] = {};  // số phần tử hiện có trong túi  int size;  public:  // Hàm tạo  Bag() {  size = 0;  }  // Thêm một phần tử vào túi  void add(int value) {  //nếu tồn tại phần tử thì tăng count lên  for (int i = 0; i < size; i++) {  if (data[i].value == value) {  data[i].count++;  return;  }  }  //nếu không thì tạo data mới  data[size].value = value;  data[size].count = 1;  size++;  }  // Xóa một phần tử khỏi túi  void remove(int value) {  for (int i = 0; i < size; i++) {  //nếu tồn tại thì giảm count đi  if (data[i].value == value) {  data[i].count--;  //count = 0 thì bỏ node data  if (data[i].count == 0) {  for (int j = i; j < size - 1; j++) {  data[j] = data[j + 1];  }  size--;  }  return;  }  }  }  // Xóa hết một phần tử khỏi túi  void removeAll(int value) {  for (int i = 0; i < size; i++) {  if (data[i].value == value) {  for (int j = i; j < size - 1; j++) {  data[j] = data[j + 1];  }  size--;  i--;  }  }  }  // Đếm số lần xuất hiện của một phần tử trong túi  int count(int value) {  for (int i = 0; i < size; i++) {  if (data[i].value == value) {  return data[i].count;  }  }  return 0;  }  // Kiểm tra hai túi có bằng nhau không  bool isBagsEqual(Bag& bag2) {  if (this->size != bag2.size) {  return false;  }  for (int i = 0; i < this->size; i++) {  if (this->count(this->data[i].value) != bag2.count(bag2.data[i].value)) {  return false;  }  }  return true;  }  // Kiểm tra một túi có là túi con của túi khác không  bool isSubsetOf(Bag other) {  for (int i = 0; i < size; i++) {  if (count(data[i].value) > other.count(data[i].value)) {  return false;  }  }  return true;  }  //gộp 2 túi với nhau  Bag bagUnion(Bag& bag2) {  Bag result;  for (int i = 0; i < this->size; i++) {  for (int j = 0; j < this->count(this->data[i].value); j++) {  result.add(this->data[i].value);  }  }  result.display();  for (int i = 0; i < bag2.size; i++) {  bool found = false;  for (int j = 0; j < result.size; j++) {  if (result.data[j].value == bag2.data[i].value) {  result.data[j].count += bag2.data[i].count;  found = true;  break;  }  }  if (!found) {  result.add(bag2.data[i].value);  }  }  return result;  }  //in các phần tử trong túi  void display() {  if (size == 0) {  cout << "Bag is empty.\n";  return;  }  cout << "Bag content (value, count):\n";  for (int i = 0; i < size; i++) {  cout << "(" << data[i].value << ", " << data[i].count << ")" << "\n";  }  }  };  int main() {  Bag bag1, bag2;  bag1.add(1);  bag1.add(2);  bag1.add(3);  bag1.add(2);  //bag1  cout << "Bag1:\n";  bag1.display();  bag2.add(2);  bag2.add(1);  bag2.add(2);  bag2.add(3);  //bag2  cout << "Bag2:\n";  bag2.display();  //lấy số lần xuất hiện của giá trị 2 trong bag1  std::cout << "Number of occurrences of 2 in bag1: " << bag1.count(2) << std::endl;  //kiểm tra 2 bang có bằng nhau  if (bag1.isBagsEqual(bag2)) {  std::cout << "bag1 and bag2 are equal" << std::endl;  }  else {  std::cout << "bag1 and bag2 are not equal" << std::endl;  }  //kiểm tra có phải subset  if (bag1.isSubsetOf(bag2)) {  std::cout << "bag1 is a subset of bag2" << std::endl;  }  else {  std::cout << "bag1 is not a subset of bag2" << std::endl;  }  //gộp 2 bag  Bag bag3 = bag1.bagUnion(bag2);  //bag2  cout << "bagUnion bag1 and bag2:\n";  bag3.display();  cout << "Remove all value 2 in bagUnion:\n";  bag3.removeAll(2);  bag3.display();  return 0;  } |

Kết quả chạy thử với hàm main() ở trên:

Text

Description automatically generated

**Cấu trúc dữ liệu túi sử dụng bảng băm:**

|  |
| --- |
| #include <iostream>  using namespace std;  struct Data {  int value;  int count;  Data\* next;  };  struct Bag {  private:  Data\* hashTable[1000] = {};  int size;  public:  //hàm khởi tạo  Bag() {  size = 0;  }  //hàm băm sử dụng thuật toán FNV  int hash(int value) {  unsigned int h = 2166136261;  char\* p = (char\*)&value;  for (int i = 0; i < sizeof(int); i++) {  h = (h \* 16777619) ^ p[i];  }  return h % 1000;  }  //thêm phần tử vào túi  void add(int value) {  int index = hash(value);  Data\* data = hashTable[index];  while (data != NULL && data->value != value) {  data = data->next;  }  if (data != NULL) {  data->count++;  }  else {  data = new Data{ value, 1, hashTable[index] };  hashTable[index] = data;  size++;  }  }  //xóa 1 phần tử ra khỏi túi  void remove(int value) {  int index = hash(value);  Data\* data = hashTable[index];  Data\* prev = NULL;  while (data != NULL && data->value != value) {  prev = data;  data = data->next;  }  if (data != NULL) {  data->count--;  if (data->count == 0) {  if (prev == NULL) {  hashTable[index] = data->next;  }  else {  prev->next = data->next;  }  delete data;  size--;  }  }  }  //xóa tất cả phần tử value có trong túi  void removeAll(int value) {  int index = hash(value);  Data\* data = hashTable[index];  Data\* prev = NULL;  while (data != NULL) {  if (data->value == value) {  if (prev == NULL) {  hashTable[index] = data->next;  }  else {  prev->next = data->next;  }  Data\* tmp = data;  data = data->next;  delete tmp;  size--;  }  else {  prev = data;  data = data->next;  }  }  }  //đếm số lân xuất hiện của phần tử trong túi  int count(int value) {  int index = hash(value);  Data\* data = hashTable[index];  while (data != NULL && data->value != value) {  data = data->next;  }  if (data != NULL) {  return data->count;  }  return 0;  }  //kiểm tra 2 túi có bắng nhau  bool isBagsEqual(Bag& bag2) {  if (size != bag2.size) {  return false;  }  for (int i = 0; i < 1000; i++) {  Data\* data1 = hashTable[i];  Data\* data2 = bag2.hashTable[i];  while (data1 != NULL && data2 != NULL) {  if (data1->value != data2->value || data1->count != data2->count) {  return false;  }  data1 = data1->next;  data2 = data2->next;  }  if (data1 != NULL || data2 != NULL) {  return false;  }  }  return true;  }  //hàm thêm phần tử hỗ trợ thêm một node data vào mảng Data  //phục vụ cho hàm bagUnion  void add2(Data& data) {  int index = hash(data.value);  Data\* existingData = hashTable[index];  Data\* prevData = NULL;  while (existingData != NULL && existingData->value != data.value) {  prevData = existingData;  existingData = existingData->next;  }  if (existingData != NULL) {  existingData->count += data.count;  }  else {  Data\* newData = new Data{ data.value, data.count, hashTable[index] };  hashTable[index] = newData;  size++;  }  }  //gộp 2 túi  Bag bagUnion(Bag& bag2) {  Bag result;  for (int i = 0; i < 1000; i++) {  Data\* data = this->hashTable[i];  while (data != NULL) {  int value = data->value;  int count = data->count;  result.add2(\*data);  data = data->next;  }  data = bag2.hashTable[i];  while (data != NULL) {  int value = data->value;  int count = data->count;  result.add2(\*data);  data = data->next;  }  }  return result;  }  //kiểm tra có là túi con  bool isSubsetOf(Bag other) {  if (other.size==0) return false;  // Lặp qua các phần tử của bag  for (int i = 0; i < 1000; i++) {  Data\* data = this->hashTable[i];  while (data != NULL) {  if (other.count(data->value) < data->count) {  return false;  }  data = data->next;  }  }  return true;  }  //in ra tất cả phần tử trong data  void display() {  cout << "Bag content (value, count):\n";  for (int i = 0; i < 1000; i++) {  Data\* data = hashTable[i];  while (data != NULL) {  cout <<"("<< data->value << ", " << data->count << ")" << " ";  data = data->next;  }  }  cout << endl;  }  };  int main() {  Bag bag1, bag2;  // Thêm phần tử vào bag1  bag1.add(1);  bag1.add(2);  bag1.add(2);  bag1.add(3);  bag1.add(3);  bag1.add(3);  bag1.add(4);  bag1.add(4);  bag1.add(4);  bag1.add(4);  // Thêm phần tử vào bag2  bag2.add(2);  bag2.add(3);  bag2.add(3);  bag2.add(4);  bag2.add(4);  bag2.add(4);  bag2.add(4);  bag2.add(5);  bag2.add(5);  cout << "Bag 1:" << endl;  bag1.display();  cout << "Bag 2:" << endl;  bag2.display();  // Kiểm tra số lần xuất hiện của phần tử trong bag1  cout << "Dem so luong phan tu 3 trong bag1: " << bag1.count(3) << endl;  // Gộp 2 bag lại với nhau  Bag bagUnion = bag1.bagUnion(bag2);  cout << "Gop bag1 va bag2:" << endl;  bagUnion.display();  // Kiểm tra tính bằng nhau giữa bag1 và bagUnion  if (bag1.isBagsEqual(bagUnion)) {  cout << "bag1 và bagUnion bang nhau." << endl;  }  else {  cout << "bag1 và bagUnion khong bang nhau." << endl;  }  //kiểm tra bag1 có la túi con của bag union  if (bag1.isSubsetOf(bagUnion)) {  cout << "bag1 la tui con cua bagUnion." << endl;  }  else {  cout << "bag1 khong la tui con cua bagUnion." << endl;  }  return 0;  } |

Chạy thử với hàm main() ở trong đoạn code:

Text

Description automatically generated

So sánh độ phức tạp trung bình 2 cách trên:

|  |  |  |
| --- | --- | --- |
| Phương thức | Dùng mảng | Dùng bảng băm |
| Thêm | O(n) | O(1) |
| Xóa | O(n) | O(1) |
| Xóa hết | O(n) | O(1) |
| Đếm số lần xuất hiên | O(n) | O(1) |
| Kiểm tra bằng nhau | O(n) | O(n) |
| Kiểm tra là túi con | O(n) | O(n) |
| Hợp hai túi | O(n\*m) | O(n\*m) |

Cấu trúc bag dùng bảng băm chạy tốt hơn so với dùng mảng ở một số thao tác.

**3.**

|  |
| --- |
| #include <iostream>  using namespace std;  struct Node {  int data;  Node\* next;  };  struct List {  Node\* head;  List() {  head = NULL;  }  void insert(int data) {  Node\* newNode = new Node;  newNode->data = data;  newNode->next = head;  this->head = newNode;  }  void printList() {  Node\* current = this->head;  cout << "DSLK: \n";  while (current != NULL) {  cout << current->data << " ";  current = current->next;  }  cout << endl;  }  int sumNeg(Node\* current) {  if (current == NULL) return 0;  int sum = 0;  if (current->data < 0) sum = current->data;  return sum + sumNeg(current->next);  }  };  int main() {  List list;  list.insert(5);  list.insert(7);  list.insert(9);  list.insert(-4);  list.insert(-2);  list.insert(-10);  list.insert(9);  list.insert(-19);  Node\* head = list.head;  list.printList();  cout <<"Tong phan tu am trong DSLK: "<< list.sumNeg(head);  return 0;  } |

Chạy thử chương trình trên với hàm main():

**Text

Description automatically generated**

**4.**

|  |
| --- |
| #include <iostream>  using namespace std;  struct Node {  int data;  Node\* left;  Node\* right;  Node(int data) {  this->data = data;  left = NULL;  right = NULL;  }  };  struct BinaryTree {  Node\* root;  BinaryTree() {  root = NULL;  }  void insert(int data) {  Node\* newNode = new Node(data);  if (root == NULL) {  root = newNode;  return;  }  Node\* current = root;  while (true) {  if (data < current->data) {  if (current->left == NULL) {  current->left = new Node(data);  return;  }  current = current->left;  }  else {  if (current->right == NULL) {  current->right = new Node(data);  return;  }  current = current->right;  }  }  }  void preOrder(Node\* node) {  if (node == NULL) return;  cout << node->data << " ";  preOrder(node->left);  preOrder(node->right);  }  int sumNeg(Node\* root) {  if (root == NULL) return 0;  int sum = 0;  if (root->data < 0) {  sum += root->data;  }  sum += sumNeg(root->left);  sum += sumNeg(root->right);  return sum;  }  };  int main() {  BinaryTree tree;  tree.insert(6);  tree.insert(3);  tree.insert(4);  tree.insert(7);  tree.insert(9);  tree.insert(-2);  tree.insert(-4);  cout << "Duyet truoc cay: \n";  tree.preOrder(tree.root);  cout << endl;  cout << "Tong cac phan tu am trong cay: ";  cout << tree.sumNeg(tree.root);  return 0;  } |

**Chạy thử chương trình với ham main():**

**Text

Description automatically generated**

**5.**

a) Cây biểu thức số học:

b) Duyệt trước: \* 1 + – 2 ! + 3 4 5

c) Duyệt sau: 1 2 3 4 + ! - + 5 \*