|  |  |  |
| --- | --- | --- |
| void selectionSort(int a[], int n)  {  for (int i = 0; i < n; i++) {  for (int j = i + 1; j < n; j++) {  if (a[j] < a[i]) {  HoanVi(a[i], a[j]);  }  }  }  } | void insertionSort(int a[], int n)  {  for (int i = 1; i < n; i++) {  for (int j = 0; j < i; j++) {  if (a[j] > a[i]) {  int temp = a[i];  for (int k = i; k > j; k--) {  a[k] = a[k - 1];  }  a[j] = temp;  }  }  }  } | void bubbleSort(int a[], int n)  {  for (int i = (n - 1); i >= 0; i--) {  for (int j = 0; j < i; j++) {  if (a[j] > a[j + 1]) {  HoanVi(a[j], a[j + 1]);  }  }  }  } |
| void hieuChinh(int A[], int n, int pos)  {  int l = (n - 1) / 2;  int vtmax;  vtmax = 2 \* pos + 1;  if ((2 \* pos + 2 < n) && (A[2 \* pos + 2] > A[2 \* pos + 1])) vtmax = 2 \* pos + 2;  if (A[vtmax] > A[pos]) {  HoanVi(A[vtmax], A[pos]);  if (vtmax < l) hieuChinh(A, n, vtmax);  }  } | void taoHeap(int A[], int n)  {  int l = (n - 1) / 2;  while (l >= 0) {  hieuChinh(A, n, l);  l--;  }  } | void heapSort(int A[], int n)  {  taoHeap(A, n);  HoanVi(A[0], A[n - 1]);  while (n > 2)  {  n--;  hieuChinh(A, n, 0);  HoanVi(A[0], A[n - 1]);  }  } |
| void mquickSort(int a[], int left, int right)  {  int pivot = (left + right) / 2;  int i = left;  int j = right;  while (i <= j)  {  while (a[i] < a[pivot]) i++;  while (a[j] > a[pivot]) j--;  if (j >= i) {  HoanVi(a[i], a[j]);  i++;  j--;  }  }  if (j > left) mquickSort(a, left, j);  if (i < right) mquickSort(a, i, right);  } | void mergeSort(int \*a, int left, int right)  {  if (right > left)  {  int mid;  mid = (left + right) / 2;  mergeSort(a, left, mid);  mergeSort(a, mid + 1, right);  Merge(a, left, mid, right);  }  } |  |

|  |  |  |
| --- | --- | --- |
| bool bruteForce(string T, string P)  {  bool kq = false;  if (T.length() < P.length()) {  cout << endl << "Khong tim thay.";  return false;  }  for (int i = 0; i <= (T.length() - P.length()); i++) {  if (cmpAB(T, P, i) == false) {  continue;  }  else {  cout << endl << "Tim thay tai vi tri: " << i + 1;  kq = true;  }  }  return kq;  } | bool rabinKarp(string T, string P)  {  bool kq = false;  //Nếu chuỗi P = ""; thì không cần tìm  if (P.length() < 1 || T.length() < 1) {  cout << endl << "Chuoi can tim la chuoi rong.";  }  else {  //Tính chuỗi P thành số  int nP = 0;  for (int i = 0; i < P.length(); i++) {  nP += (int)P[i];  }  //Tính chuỗi con đầu của T có độ dài bằng length(P)  int nT = 0;  for (int i = 0; i < P.length(); i++) {  nT += (int)T[i];  }  //Xét xem chuỗi đầu của T ở trên có giống chuỗi P không  if (nP == nT) {  if (cmpAB(T, P, 0)) {  cout << endl << "Tim thay tai vi tri: 1";  kq = true;  }  }  //Đi từ vị trí phần tử thứ 2 đến phần tử n T.length() - P.length() + 1  for (int i = 1; i < (T.length() - P.length() + 1); i++) {  nT = nT - (int)T[i - 1] + (int)T[i + P.length() - 1];  if (nP == nT) {  if (cmpAB(T, P, i)) {  cout << endl << "Tim thay tai vi tri: " << i + 1;  kq = true;  }  }  }  }  if (kq == false) {  cout << endl << "Khong tim thay.";  }  return kq;  } | int\* generateNEXT(string P)  {  int \*NEXT = new int[P.length()];  string P1, P2;  NEXT[0] = -1;  \*(NEXT + 1) = 0;  for (int i = 2; i < P.length(); i++) {  for (int j = 0; j < (i - 1); j++) {  P1 += P[j];  }  for (int j = 1; j < i; j++) {  P2 += P[j];  }  while (P1 != P2)  {  //không khớp thì bới chuỗi con đầu ký tự cuối cùng, bớt chuỗi con sau ký tự đầu tiên.  P1.pop\_back();  P2.erase(0, 1);  }  \*(NEXT + i) = P1.length();  P1.clear();  P2.clear();  }  return NEXT;  }  bool morrisPrutt(string T, string P)  {  bool kq = false;  if (T.length() < P.length()) {  cout << endl << "Khong tim thay.";  return false;  }  int\* NEXT = generateNEXT(P);  int i = 0, j = 0;  while (i <= (T.length() - P.length()))  {  j = 0;  while (T[i + j] == P[j]) {  j++;  }  if (j == P.length()) {  cout << endl << "Tim thay tai vi tri: " << i + 1;  kq = true;  i++;  }  else {  i = i + j - NEXT[j];  }  }  return kq;  } |

|  |  |  |
| --- | --- | --- |
| void inOrderTravel(BST T)  {  //Nếu node góc là node lá thì in node góc ra màn hình  if (T.root->pLeft == NULL && T.root->pRight == NULL) std::cout << T.root->data << " ";  else {  //Nếu node bên trái khác null thì inorder node bên trái  if (T.root->pLeft != NULL) {  BST temp;  temp.root = T.root->pLeft;  inOrderTravel(temp);  }  //In ra node ở giữa  std::cout << T.root->data << " ";  //Nếu node bên phải khác null thì inorder node bên phải  if (T.root->pRight != NULL) {  BST temp;  temp.root = T.root->pRight;  inOrderTravel(temp);  }  }  } | //Duyệt trước - đi từ góc  void preOrderTravel(BST T)  {  //Nếu cây khác rỗng thì in root ra  if (isEmpty(T) == false) {  std::cout << T.root->data << " ";  BST temp;  //Duyệt trước cây con bên trái  temp.root = T.root->pLeft;  preOrderTravel(temp);  //Duyệt trước cây con bên phải  temp.root = T.root->pRight;  preOrderTravel(temp);  }  } | //Duyệt sau - đi từ lá góc nằm sau cùng  void postOrderTravel(BST T)  {  //Nếu cây khác rỗng  if (isEmpty(T) != true) {  //Nếu T có cây con trái thì duyệt sau cây con trái  if (T.root->pLeft != NULL) {  BST temp;  temp.root = T.root->pLeft;  postOrderTravel(temp);  }  //Nếu T có cây con phải thì duyệt sau cây con phải  if (T.root->pRight != NULL) {  BST temp;  temp.root = T.root->pRight;  postOrderTravel(temp);  }  //In root ra  std::cout << T.root->data << " ";  }  } |