|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA  **KHOA CÔNG NGHỆ THÔNG TIN** |

**ĐỀ THI VÀ BÀI LÀM**

Tên học phần: Toán ứng dụng CNTT

Mã học phần: Hình thức thi: *Tự luận có giám sát*

Đề số: **Đ0004** Thời gian làm bài: 90 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên:** Nguyễn Hữu Hoàng Luân **Lớp**: 23T\_NHAT1 **MSSV**: 102230028

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV\_HọTên.pdf và nộp bài thông qua MS Teams.

***Câu 1*** (*2.0 điểm*): Viết chương trình (có sử dụng hàm) thực hiện công việc sau:  
 a) *(1.0 điểm)* Tìm số M là ước số nguyên tố lớn nhất của N, biết rằng N=95777.

|  |
| --- |
| **# Trả lời: Dán code bên dưới:**  #include<iostream>  using namespace std;  vector<bool> SieveOfEratosthenes(const int& n) {  vector<bool> prime(n + 1, true);  prime[0] = prime[1] = false;  for(int i = 2; i \* i < n; i++) {  if(prime[i]) {  for (int j = i \* i; j <= n; j+=i) {  prime[j] = false;  }  }  }  return prime;  }  void PrimeFactors(const long long& n, vector <int>& factors, vector <int>& counts, const vector<bool>& prime)  {  long long number = n;  for(int i = 2; i <= n/2; i++) {  if(prime[i]) {  int count = 0;  while (number % i == 0)  {  count++;  number = number / i;  }  if (count > 0) {  factors.push\_back(i);  counts.push\_back(count);  }  }  }  }  int main() {  long long n;  cout << "Enter a number: "; cin >> n;  vector<bool> prime = SieveOfEratosthenes(n);  vector<int> factors, counts;  PrimeFactors(n, factors, counts, prime);  long long maxPrimeFactor = factors[factors.size() - 1];  cout << "Max prime factor of " << n << " is: " << maxPrimeFactor << endl;  }  **# Trả lời: Dán kết quả thực thi vào bên dưới:** |

*b) (1.0 điểm)* Tìm số lượng các số hoàn hảo nhỏ hơn M, liệt kê và tính tổng của chúng.

|  |
| --- |
| **# Trả lời: Dán code vào bên dưới:**  #include <iostream>  #include <vector>  #include <cmath>  using namespace std;  bool isPerfectNumber(long long n) {  if (n <= 1) return false;    long long sum = 1;    for (long long i = 2; i <= sqrt(n); i++) {  if (n % i == 0) {  sum += i;  if (i != n / i) {  sum += n / i;  }  }  }    return (sum == n);  }  bool isPerfectNumberFormula(long long n, vector<long long>& knownPerfects) {  for (long long perfect : knownPerfects) {  if (n == perfect) return true;  }  return false;  }  void findPerfectNumbersLessThanM(long long M) {  vector<long long> perfectNumbers = {6, 28, 496, 8128, 33550336, 8589869056, 137438691328};    vector<long long> result;  long long sum = 0;    for (long long perfect : perfectNumbers) {  if (perfect < M) {  result.push\_back(perfect);  sum += perfect;  }  }    cout << "Số lượng số hoàn hảo nhỏ hơn " << M << " là: " << result.size() << endl;    cout << "Các số hoàn hảo nhỏ hơn " << M << " là: ";  for (size\_t i = 0; i < result.size(); i++) {  cout << result[i];  if (i < result.size() - 1) cout << ", ";  }  cout << endl;    cout << "Tổng các số hoàn hảo nhỏ hơn " << M << " là: " << sum << endl;  }  int main() {  long long M;  cout << "Nhập số M: ";  cin >> M;    findPerfectNumbersLessThanM(M);    return 0;  }  **# Trả lời: Dán kết quả thực thi vào bên dưới:** |

***Câu 2*** (*2.0 điểm*): Cho ma trận A. Viết chương trình (có sử dụng hàm) thực hiện phân rã ma trận A bằng phương pháp SVD.

|  |
| --- |
| **# Trả lời:** **Dán code vào bên dưới**  #include <iostream>  #include "eigen/Eigen/Dense"  #include <iomanip>  #include <cmath>  using namespace std;  using namespace Eigen;  const double EPS = 1e-5;  int count = 0;  void swap(double& a, double& b);  void display(double A[][10], int row, int col);  void transpose(double A[][10], double At[][10], int rows, int cols);  void multiplyMatrices(MatrixXd& S, double A[][10], double B[][10], int row1, int col1, int col2);  void getEigenValuesAndVectors(MatrixXd S, MatrixXd& lambda, MatrixXd& vector);  void calculateMatrixS(MatrixXd lambda, double sigma[][10], int rows, int cols);  void calculateMatrixU(MatrixXd lambda, MatrixXd vector, double U[][10], double A[][10], int rows, int cols);  void calculateMatrixV(MatrixXd vector, double V[][10]);  void gramSchmidt(double U[][10], int rows, int cols);  void rref(MatrixXd& U\_matrix);  int main()  {  int rows, cols;  double A[10][10], At[10][10];  cout << "Enter Matrix Dimensions:" << endl;  cout << "Rows: ";  cin >> rows;  cout << "Columns: ";  cin >> cols;  cout << "Enter Matrix Elements:" << endl;  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  cout << "A[" << i + 1 << "][" << j + 1 << "] = ";  cin >> A[i][j];  }  }  MatrixXd S(cols, cols), lambda(cols, 1), vector(cols, cols);  transpose(A, At, rows, cols);  multiplyMatrices(S, At, A, cols, rows, cols);  MatrixXd S\_copy = S;  rref(S\_copy);  getEigenValuesAndVectors(S, lambda, vector);  double sigma[10][10], U[10][10], V[10][10];  calculateMatrixU(lambda, vector, U, A, rows, cols);  cout << "Completed Gram-Schmidt Process..." << endl;  MatrixXd null\_space(cols, 1);  for (int i = 0; i < cols; i++)  {  null\_space(i, 0) = S\_copy(i, cols - 1);  }  cout << "Matrix U:" << endl;  display(U, rows, rows);  calculateMatrixS(lambda, sigma, rows, cols);  cout << "Matrix Sigma:" << endl;  display(sigma, rows, cols);  calculateMatrixV(vector, V);  cout << "Matrix V^T:" << endl;  double VT[10][10];  transpose(V, VT, cols, cols);  display(VT, cols, cols);  MatrixXd U\_matrix(rows, rows), V\_matrix(cols, cols), S\_matrix(rows, cols);  for (int i = 0; i < rows; i++)  for (int j = 0; j < rows; j++)  U\_matrix(i, j) = U[i][j];  for (int i = 0; i < cols; i++)  for (int j = 0; j < cols; j++)  V\_matrix(i, j) = V[i][j];  for (int i = 0; i < rows; i++)  for (int j = 0; j < cols; j++)  S\_matrix(i, j) = sigma[i][j];  MatrixXd A\_matrix = U\_matrix \* S\_matrix \* V\_matrix;  cout << "Reconstructed Matrix A:" << endl;  display(A, rows, cols);  return 0;  }  void rref(MatrixXd& U\_matrix)  {  int rows = U\_matrix.rows();  int cols = U\_matrix.cols();  int lead = 0;  for (int r = 0; r < rows; ++r)  {  if (lead >= cols)  return;  int i = r;  while (U\_matrix(i, lead) == 0)  {  ++i;  if (i == rows)  {  i = r;  ++lead;  if (lead == cols)  return;  }  }  U\_matrix.row(i).swap(U\_matrix.row(r));  U\_matrix.row(r) /= U\_matrix(r, lead);  for (int i = 0; i < rows; ++i)  {  if (i != r)  {  U\_matrix.row(i) -= U\_matrix.row(r) \* U\_matrix(i, lead);  }  }  ++lead;  }  }  void swap(double& a, double& b)  {  double temp = a;  a = b;  b = temp;  }  void display(double A[][10], int row, int col)  {  for (int i = 0; i < row; i++)  {  for (int j = 0; j < col; j++)  cout << setw(12) << setprecision(4) << A[i][j];  cout << endl;  }  }  void transpose(double A[][10], double At[][10], int rows, int cols)  {  for (int i = 0; i < rows; i++)  for (int j = 0; j < cols; j++)  At[j][i] = A[i][j];  }  void multiplyMatrices(MatrixXd& S, double A[][10], double B[][10], int row1, int col1, int col2)  {  for (int i = 0; i < row1; i++)  {  for (int j = 0; j < col2; j++)  {  S(i, j) = 0;  for (int k = 0; k < col1; k++)  {  S(i, j) += A[i][k] \* B[k][j];  }  }  }  }  void getEigenValuesAndVectors(MatrixXd S, MatrixXd& lambda, MatrixXd& vector)  {  SelfAdjointEigenSolver<Eigen::MatrixXd> eigensolver(S);  vector = eigensolver.eigenvectors();  lambda = eigensolver.eigenvalues();  int k = lambda.rows();  int l = vector.rows();  for (int i = 0; i < k; i++)  {  if (lambda(i, 0) < EPS )  {  lambda(i, 0) = 0;  ::count++;  }  }  for (int i = 0; i < k; i++)  {  for (int j = i + 1; j < k; j++)  {  if (lambda(j, 0) > lambda(i, 0))  {  swap(lambda(j, 0), lambda(i, 0));  for (int h = 0; h < l; h++)  swap(vector(h, i), vector(h, j));  }  }  }  }  void calculateMatrixU(MatrixXd lambda, MatrixXd vector, double U[][10], double A[][10], int rows, int cols)  {  MatrixXd ui(rows, 1);  double Vi[10][10];  for (int i = 0; i < cols; i++)  {  for (int j = 0; j < cols; j++)  {  Vi[j][0] = vector(j, i);  }  multiplyMatrices(ui, A, Vi, rows, cols, 1);  for (int k = 0; k < rows; k++)  {  if (lambda(i, 0) != 0)  {  U[k][i] = (1 / sqrt(lambda(i, 0))) \* ui(k, 0);  }  else  {  U[k][i] = 0;  }  }  }  }  void gramSchmidt(double U[][10], int rows, int cols)  {  for (int i = cols; i < rows; i++)  {  for (int j = 0; j < rows; j++)  {  U[j][i] = (i == j) ? 1 : 0;  }  for (int k = 0; k < i; k++)  {  double dot\_product = 0;  for (int j = 0; j < rows; j++)  {  dot\_product += U[j][i] \* U[j][k];  }  for (int j = 0; j < rows; j++)  {  U[j][i] -= dot\_product \* U[j][k];  }  }  double norm = 0;  for (int j = 0; j < rows; j++)  {  norm += U[j][i] \* U[j][i];  }  norm = sqrt(norm);  for (int j = 0; j < rows; j++)  {  U[j][i] /= norm;  }  }  }  void calculateMatrixV(MatrixXd vector, double V[][10])  {  int row = vector.rows(), col = vector.cols();  for (int i = 0; i < row; i++)  {  for (int j = 0; j < col; j++)  {  V[i][j] = vector(i, j);  }  }  }  void calculateMatrixS(MatrixXd lambda, double sigma[][10], int rows, int cols)  {  int k = 0;  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  if (i == j && lambda(k, 0) != 0)  {  sigma[i][j] = sqrt(lambda(k, 0));  k++;  }  else  {  sigma[i][j] = 0;  }  }  }  }  **# Trả lời:** **Dán kết quả thực thi** vào bên dưới biết rằng , sai số . |

***Câu 3*** (*3.0 điểm*): Cho mười lăm điểm trong không gian Oxy như sau: {3,5}, {6,8},{9,10}, {11,2},{4,9},{7,8},{9,9},{10,3},{11,15},{34,5},{32,1},{25,10},{18,8},{7,18}*,*{*10,14*}.

1. *(1.0 điểm) Mô tả thuật toán xác định bao lồi của tập điểm đã cho*

|  |
| --- |
| **# Trả lời:** **dán sơ đồ khối hoặc mã giả:**  **Input**: Tập điểm P = {p1, p2,..,p3}  **Output**: Danh sách các điểm tạo thành bao lồi theo thứ tự ngược chiều kim đồng hồ  **Thuật toán GrahamScan(P):**  1. Nếu số điểm < 3, trả về P (bao lồi không xác định)  2. Tìm điểm có tung độ nhỏ nhất (nếu bằng thì hoành độ nhỏ nhất) → gọi là điểm gốc P0  3. Sắp xếp các điểm còn lại theo góc tạo bởi vector (P0 → Pi) với trục hoành (góc cực)  (nếu hai điểm có cùng góc, giữ lại điểm xa hơn P0)  4. Khởi tạo stack S, push P0, điểm đầu tiên, và điểm thứ hai sau sắp xếp vào S  5. Duyệt i từ điểm thứ ba đến hết:  while S có ít nhất 2 điểm và hướng rẽ tại 3 điểm kế tiếp không là rẽ trái:  pop điểm trên cùng khỏi S  push điểm Pi vào S  6. Stack S chứa các điểm của bao lồi theo thứ tự ngược chiều kim đồng hồ  Trả về S |

1. *(1.0 điểm)* Xác định bao lồi và cạnh nhỏ nhất của đa giác lồi vừa tìm được

|  |
| --- |
| **# Trả lời:** **Dán code bên dưới:**  #include<iostream>  #include<vector>  #include<algorithm>  #include<cmath>  #include<iomanip>  #include<climits>  #include<cfloat>  using namespace std;  typedef long long ll;  struct Point{  ll x, y;  Point() : x(0), y(0) {}  Point(ll x, ll y) : x(x), y(y) {}    ll dist(const Point& other)const {  return (x - other.x) \* (x - other.x) + (y - other.y) \* (y - other.y);  }  };  struct Vector{  ll x, y;  Vector() : x(0), y(0) {}  Vector(ll x, ll y) : x(x), y(y) {}  ll crossProduct(const Vector& other){  return x \* other.y - y \* other.x;  }  };  Vector operator-(const Point& a, const Point& b){  return Vector(a.x - b.x, a.y - b.y);  }  int orientation(const Point& a, const Point& b, const Point& c){  Vector ab = b - a;  Vector ac = c - a;  ll orient = ab.crossProduct(ac);  if(orient == 0) return 0;  return (orient > 0) ? 1 : 2;  }  bool collinear(const Point& a, const Point& b, const Point& c){  return orientation(a, b, c) == 0;  }  bool cw(const Point& a, const Point& b, const Point& c, bool include\_collinear = false){  int orient = orientation(a, b, c);  return (orient == 2 || (include\_collinear && orient == 0));  }  bool cmd(const Point& a, const Point& b, const Point& P){  int orient = orientation(P, a, b);  if(orient == 0) return a.dist(P) < b.dist(P);  return (orient == 1);  }  double findMinEdge(const vector<Point>& hull){  if(hull.size() < 2) return 0.0;    double minEdge = sqrt((double)hull[0].dist(hull[hull.size() - 1]));  for(int i = 0; i < hull.size() - 1; i++){  double edge = sqrt((double)hull[i].dist(hull[i + 1]));  minEdge = min(minEdge, edge);  }  return minEdge;  }  double distanceOnHull(const vector<Point>& hull, bool mode){  double Distance = sqrt(hull[0].dist(hull[hull.size() - 1]));  if(mode){  for(int i = 0; i < hull.size() - 1; i++){  Distance = max(Distance, sqrt((double)hull[i].dist(hull[i + 1])));  }  }  else{  for(int i = 0; i < hull.size() - 1; i++){  Distance = min(Distance, sqrt((double)hull[i].dist(hull[i + 1])));  }  }  return Distance;  }  vector <Point> convexHull(const ll& n, vector<Point>& points, bool include\_collinear = false){  if(n < 3) return points;  Point P(INT\_MAX, INT\_MAX);  for(auto& p : points){  if(p.y < P.y || (p.y == P.y && p.x < P.x))  P = p;  }    sort(points.begin(), points.end(), [&](const Point& a, const Point& b){  return cmd(a, b, P);  });  if(include\_collinear){  int i = n - 1;  while(i >= 0 && collinear(points[0], points[i], P)) i--;  reverse(points.begin() + i + 1, points.end());  }  vector<Point> hull;  for(int i = 0; i < n; i++){  while(hull.size() >= 2 && cw(hull[hull.size() - 2], hull[hull.size() - 1], points[i], include\_collinear)){  hull.pop\_back();  }  hull.push\_back(points[i]);  }  return hull;  }  double convexHullArea(const vector<Point>& hull){  double area = 0;  for(int i = 0; i < hull.size(); i++){  int j = (i + 1) % hull.size();  area += (hull[i].x \* hull[j].y) - (hull[j].x \* hull[i].y);  }  return abs(area) / 2.0;  }  void Output(const vector<Point>& hull){  cout << "\n=== BAO LOI CUA TAP DIEM ===" << endl;  cout << "Cac diem thuoc bao loi (theo thu tu nguoc chieu kim dong ho): " << endl;  for (auto p: hull)  cout << "(" << p.x << ", " << p.y << ")" << endl;    cout << fixed << setprecision(2);  cout << "\n=== CANH NHO NHAT CUA BAO LOI ===" << endl;  double minEdge = findMinEdge(hull);  cout << "Do dai canh nho nhat: " << minEdge << endl;  }  int main(){  cout << "Nhap so diem: ";  int n;  cin >> n;    if(n < 3){  cout << "Khong du diem de tao thanh da giac (can it nhat 3 diem)" << endl;  return 0;  }    vector<Point> points(n);  cout << "Nhap toa do cac diem (x y):" << endl;  for(int i = 0; i < n; i++){  cout << "Diem " << i+1 << ": ";  cin >> points[i].x >> points[i].y;  }  // Tìm bao lồi  vector<Point> hull = convexHull(n, points);    // In kết quả  Output(hull);    return 0;  }  **# Trả lời:** **Dán kết quả thực thi vào bên dưới:** |

c)  *(1.0 điểm)* Xác định số lượng các điểm nằm bên trong bao lồi và liệt kê chúng

|  |
| --- |
| **# Trả lời:** **Dán code bên dưới:**  #include<iostream>  #include<vector>  #include<algorithm>  #include<climits>  using namespace std;  typedef long long ll;  struct Point{  ll x, y;  Point() : x(0), y(0) {}  Point(ll x, ll y) : x(x), y(y) {}    ll dist(const Point& other)const {  return (x - other.x) \* (x - other.x) + (y - other.y) \* (y - other.y);  }  };  struct Vector{  ll x, y;  Vector() : x(0), y(0) {}  Vector(ll x, ll y) : x(x), y(y) {}  ll crossProduct(const Vector& other){  return x \* other.y - y \* other.x;  }  };  Vector operator-(const Point& a, const Point& b){  return Vector(a.x - b.x, a.y - b.y);  }  int orientation(const Point& a, const Point& b, const Point& c){  Vector ab = b - a;  Vector ac = c - a;  ll orient = ab.crossProduct(ac);  if(orient == 0) return 0;  return (orient > 0) ? 1 : 2;  }  bool collinear(const Point& a, const Point& b, const Point& c){  return orientation(a, b, c) == 0;  }  bool cw(const Point& a, const Point& b, const Point& c, bool include\_collinear = false){  int orient = orientation(a, b, c);  return (orient == 2 || (include\_collinear && orient == 0));  }  bool cmd(const Point& a, const Point& b, const Point& P){  int orient = orientation(P, a, b);  if(orient == 0) return a.dist(P) < b.dist(P);  return (orient == 1);  }  bool isPointInsideConvexHull(const Point& point, const vector<Point>& hull){  int n = hull.size();  if(n < 3) return false;    for(const auto& h : hull){  if(point.x == h.x && point.y == h.y){  return false;  }  }    for(int i = 0; i < n; i++){  int j = (i + 1) % n;  if(orientation(hull[i], hull[j], point) == 2){  return false;  }  }  return true;  }  vector<Point> findPointsInsideHull(const vector<Point>& allPoints, const vector<Point>& hull){  vector<Point> insidePoints;  for(const auto& point : allPoints){  if(isPointInsideConvexHull(point, hull)){  insidePoints.push\_back(point);  }  }  return insidePoints;  }  vector <Point> convexHull(const ll& n, vector<Point>& points, bool include\_collinear = false){  if(n < 3) return points;  Point P(INT\_MAX, INT\_MAX);  for(auto& p : points){  if(p.y < P.y || (p.y == P.y && p.x < P.x))  P = p;  }    sort(points.begin(), points.end(), [&](const Point& a, const Point& b){  return cmd(a, b, P);  });  if(include\_collinear){  int i = n - 1;  while(i >= 0 && collinear(points[0], points[i], P)) i--;  reverse(points.begin() + i + 1, points.end());  }  vector<Point> hull;  for(int i = 0; i < n; i++){  while(hull.size() >= 2 && cw(hull[hull.size() - 2], hull[hull.size() - 1], points[i], include\_collinear)){  hull.pop\_back();  }  hull.push\_back(points[i]);  }  return hull;  }    void Output(const vector<Point>& allPoints, const vector<Point>& hull){  vector<Point> insidePoints = findPointsInsideHull(allPoints, hull);    cout << "So luong diem nam trong bao loi: " << insidePoints.size() << endl;    if(insidePoints.size() > 0){  cout << "Diem nam trong bao loi:" << endl;  for(int i = 0; i < insidePoints.size(); i++){  cout << "(" << insidePoints[i].x << ", " << insidePoints[i].y << ")" << endl;  }  }  }  int main(){  int n;  cout << "Nhap so luong diem: ";  cin >> n;    if(n < 3){  cout << "Can it nhat 3 diem " << endl;  return 0;  }    vector<Point> points(n);  for(int i = 0; i < n; i++){  cout << "Diem " << i + 1 << ": ";  cin >> points[i].x >> points[i].y;  }  vector<Point> hull = convexHull(n, points);    Output(points, hull);    return 0;  }  **# Trả lời:** **Dán kết quả thực thi vào bên dưới:** |

***Câu 4*** (*2.0 điểm*): Cho hàm số 

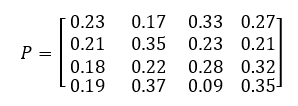
1. *(1.0 điểm) Trình bày thuật toán tối ưu hàm số đã cho* sử dụng phương pháp *gradient descent với momentum*, biết rằng tham số học (learning rate)  , hệ số động lượng là .

|  |
| --- |
| **# Trả lời: dán sơ đồ khối hoặc mã giả:** |

1. *(1.0 điểm)* Viết chương trình (có dùng hàm) tính giá trị bé nhất của f(x) sử dụng phương pháp *gradient descent với momentum* với số bước lặp *N* và sai số .

|  |
| --- |
| **# Trả lời**: **Dán code vào bên dưới:**  **# Trả lời**: **Dán kết quả thực thi** với điểm khởi , tham số học học (*learning rate*) , hệ số động lượng (*momentum coefficient*) là , số bước lặp và sai số : |

***Câu 5*** (*1.0 điểm*): Một hệ thống có chế độ làm việc ở mỗi giai đoạn vận hành chỉ với các trạng thái từ 1 đến 4. Chế độ làm việc của hệ thống này được mô tả bằng ma trận chuyển như sau:



a) (*0.5 điểm*) Vẽ đồ thị biểu diễn chuỗi Markov tương ứng đã cho

|  |
| --- |
| **# Trả lời:** **Dán kết quả vào bên dưới** |

b) (*0.5 điểm*) Giả sử rằng hệ thống bắt đầu học ở trạng thái 2. Tính xác suất hệ thống làm việc ở trạng thái 4 *sau hai và bốn bước thời gian vận hành*.

|  |
| --- |
| **# Trả lời**: **Dán kết quả tính toán vào bên dưới:** |

Đà Nẵng, ngày 4 tháng 12 năm 2024

|  |  |
| --- | --- |
| **GIẢNG VIÊN BIÊN SOẠN ĐỀ THI** | **KHOA CÔNG NGHỆ THÔNG TIN**  **( đã duyệt)** |
|  |  |