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: Đ0003 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 Phúc………**Lớp**:……21TCLC_DT3……**MSSV**:……102210225……..

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

<u>Câu 1</u> (*2 điểm*): Cho M=50 và N=500. Viết chương trình (có sử dụng hàm) thực hiện:

a) *(1.0 điểm)* Tìm số lượng các số nguyên tố nằm trong khoảng M đến N, liệt kê và tính tổng của chúng.

 # Trả lời: Dán code bên dưới:

```
#include <stdio.h>
int is_prime(int num) {
   if (num < 2) {
      return 0; // False
   }
   for (int i = 2; i * i <= num; ++i) {
      if (num % i == 0) {
         return 0; // False
      }
   }
   return 1; // True
}
void find_primes_and_sum(int M, int N) {
   int count = 0;
```

```
    int sum = 0;
    printf("Cac so nguyen to tu %d den %d la:\n", M, N);
    for (int i = M; i <= N; ++i) {
        if (is_prime(i)) {
            printf("%d ", i);
            count++;
            sum += i;
        }
    }
    printf("\nTong cua cac so nguyen to: %d\n", sum);
    printf("So luong cac so nguyen to: %d\n", count);
}

int main() {
    int M = 50;
    int N = 500;
    find_primes_and_sum(M, N);
    return 0;
}
```

 # Trả lời: Dán kết quả thực thi vào bên dưới:

```
Cac so nguyen to tu 50 den 500 la:
53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 22
3 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 39
7 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499
Tong cua cac so nguyen to: 21208
So luong cac so nguyen to: 80
```

b) *(1.0 điểm) Tìm số nào gần với số 300 nhất* Trong các số nguyên tố vừa tìm được ở ý a)

 # Trả lời: Dán code bên dưới:

```
#include <stdio.h>
#include <stdlib.h>
int is_prime(int num) {
    if (num < 2) {
```

```c
        return 0;
    }
    for (int i = 2; i * i <= num; ++i) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}
void find_primes_and_sum(int M, int N) {
    int count = 0;
    int sum = 0;
    int nearest_prime = 0;
    int min_difference = abs(300 - M);

    printf("Cac so nguyen to tu %d den %d la:\n", M, N);
    for (int i = M; i <= N; ++i) {
        if (is_prime(i)) {
            printf("%d ", i);
            count++;
            sum += i;
            int difference = abs(300 - i);
            if (difference < min_difference) {
                min_difference = difference;
                nearest_prime = i;
            }
        }
    }

    printf("\nTong cua cac so nguyen to: %d\n", sum);
```

```
    printf("So luong cac so nguyen to: %d\n", count);
    printf("So nguyen to gan nhat voi 300: %d\n", nearest_prime);
}

int main() {
    int M = 50;
    int N = 500;

    find_primes_and_sum(M, N);

    return 0;
}
```

 # Trả lời: Dán kết quả thực thi vào bên dưới:

```
Cac so nguyen to tu 50 den 500 la:
53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 22
3 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 39
7 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499
Tong cua cac so nguyen to: 21208
So luong cac so nguyen to: 80
So nguyen to gan nhat voi 300: 293
```

Câu 2 (*3 điểm*): Cho ma trận A. Viết chương trình (có sử dụng hàm) phân rã ma trận A bằng phương pháp SVD.

 # Trả lời: Dán code vào bên dưới

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <Eigen/Eigenvalues>
using Eigen::MatrixXd;
using std::cout;
using std::cin;

const int _SIZE_MAX = 10;
const double eps = 1e-7;
```

```
typedef double matrix[_SIZE_MAX][_SIZE_MAX];

void enterMatrix(matrix mat, int& rows, int& cols) {
	cout << "Enter number of rows of the matrix: ";
	cin >> rows;

	cout << "Enter number of columns of the matrix: ";
	cin >> cols;

	cout << "Enter the matrix " << rows << " x " << cols << ":\n";
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			cin >> mat[i][j];
		}
	}
}

void printMatrix(matrix mat, const int rows, const int cols) {
	int i, j;
	for (i = 0; i < rows; i++) {
		for (j = 0; j < cols; j++) {
			cout << std::setw(8) << std::setprecision(5) <<  mat[i][j] << " ";
		}
		cout << "\n";
	}
}

void transpose(matrix In, const int rows, const int cols, matrix result) {
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
```

```
                result[j][i] = In[i][j];
            }
    }
}

void getColumns(matrix In, const int size, int index, matrix result) {
    for (int i = 0; i < size; i++) {
        result[i][0] = In[i][index];
    }
}

void setColumns(matrix In, const int size, int index, matrix result) {
    for (int i = 0; i < size; i++) {
        result[i][index] = In[i][0];
    }
}

void multiplyMatrices(matrix A, matrix B, const int rows1, const int cols1, const int cols2,
matrix result) {
    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < cols1; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void scaleMatrix(matrix In, const int rows, const int cols, const double scale, matrix result) {
    for (int i = 0; i < rows; i++) {
```

```
		for (int j = 0; j < cols; j++) {
			result[i][j] = scale * In[i][j];
		}
	}
}

void eigen(matrix In, const int size, double resEigenvalues[], matrix resEigenvectors) {
	MatrixXd m(size, size);
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			m(i, j) = In[i][j];
		}
	}

	Eigen::EigenSolver<Eigen::MatrixXd> solver(m);
	Eigen::VectorXd eigenvalues = solver.eigenvalues().real();
	Eigen::MatrixXd eigenvectors = solver.eigenvectors().real();

	for (int i = 0; i < eigenvalues.size(); i++) {
		resEigenvalues[i] = eigenvalues[i];
		if (fabs(resEigenvalues[i]) <= eps)
			resEigenvalues[i] = 0;
	}

	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			resEigenvectors[i][j] = eigenvectors(i, j);
			if (fabs(resEigenvectors[i][j]) <= eps)
				resEigenvectors[i][j] = 0;
		}
```

```
    }
}

void sigmaMatrix(double eigenvalues[], const int rows, const int cols, matrix result) {

    // values on init square matrix
    int num = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (i == j) {
                while (eigenvalues[num] == 0) num++;
                result[i][j] = sqrt(eigenvalues[num++]);
            }

            else
                result[i][j] = 0;
        }
    }
}

void leftSingularMatrix(matrix A, const int rows, const int cols, double eigenvalues[], matrix
eigenVectors, matrix result) {
    matrix tmp, ans;
    int pos = 0;
    for (int i = 0; i < cols; i++) {
        if (eigenvalues[i] == 0) continue;

        matrix vctCol;
        getColumns(eigenVectors, cols, i, vctCol);
        multiplyMatrices(A, vctCol, rows, cols, 1, tmp);
        scaleMatrix(tmp, rows, 1, 1 / sqrt(eigenvalues[i]), ans);
```

```
            setColumns(ans, rows, pos, result);
            pos++;
        }
}

void singularValueDecomposition(matrix A, const int rows, const int cols) {

        matrix AT;
        transpose(A, rows, cols, AT);

        matrix mRes;
        multiplyMatrices(AT, A, cols, rows, cols, mRes);
        cout << "\nAT * A = \n";
        printMatrix(mRes, cols, cols);

        double eigenValues[cols];
        matrix eigenVectors;
        int num = 0;
        eigen(mRes, cols, eigenValues, eigenVectors);
        cout << "\nEigenvalues: ";
        for (int i = 0; i < cols; i++) cout << eigenValues[i] << " ";
        cout << "\n\nEigenvectors matrix P = \n";
        printMatrix(eigenVectors, cols, cols);

        matrix VT;
        transpose(eigenVectors, cols, cols, VT);

        matrix sigma;
        sigmaMatrix(eigenValues, rows, cols, sigma);
```

```
	matrix leftSingular;
	leftSingularMatrix(A, rows, cols, eigenValues, eigenVectors, leftSingular);

	// Conclusion

	cout << "\nSolution: A = U * sigma * VT, with:\n";

	cout << "\nU = \n";
	printMatrix(leftSingular, rows, rows);

	cout << "\nsigma = \n";
	printMatrix(sigma, rows, cols);

	cout << "\nVT = \n";
	printMatrix(VT, cols, cols);
}

int main()
{
	int r, c;
	matrix A;
	enterMatrix(A, r, c);
	singularValueDecomposition(A, r, c);}
```

 # Trả lời: Dán kết quả thực thi vào bên dưới biết rằng $A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 2 & 5 \end{bmatrix}$, sai số $\varepsilon = 10^{-5}$.

```
U =
[[-0.46094  0.78795  0.40825]
 [-0.57039  0.08937 -0.8165 ]
 [-0.67985 -0.60922  0.40825]]

Sigma =
[[8.03338 0.      0.     ]
 [0.      0.68181 0.     ]
 [0.      0.      0.     ]]

VT =
[[-0.21301 -0.42602 -0.87928]
 [ 0.39323  0.78645 -0.4763 ]
 [-0.89443  0.44721  0.      ]]
```

<u>Câu 3</u> (*2 điểm*): Cho mười điểm trong không gian Oxy như sau: (2, 5); (3,7); (4,3); (2,9); (6,12); (7,16); (8,3); (9,8); (10,7); (11,12)

a) *(1.0 điểm) Mô tả thuật toán xác định bao lồi*

 # Trả lời: dán sơ đồ khối hoặc mã giả

```
function GrahamScan(D):
    # D là danh sách các điểm đầu vào
    # Bước 1: Chọn điểm bắt đầu
    P0 = Chọn điểm có tọa độ y nhỏ nhất, nếu tie, chọn x nhỏ nhất
    # Bước 2: Sắp xếp các điểm theo góc so với P0
    Sắp xếp các điểm còn lại của D theo góc tạo bởi đường thẳng từ P0
    # Bước 3: Khởi tạo ngăn xếp với ba điểm đầu
    stack = [P0, điểm đầu tiên sau P0 sau khi sắp xếp, điểm thứ hai sau P0 sau khi sắp xếp]
    # Bước 4: Duyệt qua các điểm còn lại
    for i từ 3 đến số lượng điểm trong D:
        while stack có nhiều hơn 1 điểm và thêm điểm tiếp theo làm phá vỡ tính lồi:
            Loại bỏ điểm đỉnh cùng của stack
        # Thêm điểm tiếp theo vào stack
        Đẩy điểm D[i] vào stack
    # Bước 5: Kết quả là ngăn xếp chứa các điểm trên bao lồi
    return stack
```

b) *(1.0 điểm)* Viết hàm xác định bao lồi và tính diện tích bao lồi tìm được.

 # Trả lời: Dán code bên dưới:

```
#include <bits/stdc++.h>

#include <set>

#define llu long long int

#define RIGHT_TURN -1  // CW

#define LEFT_TURN 1  // CCW

#define COLLINEAR 0  // Collinear

using namespace std;

struct Point {

	llu x, y;

	bool operator<(Point p)

	{

		return x < p.x || (x == p.x && y < p.y);

	}


	bool operator>(Point p)

	{

		return x > p.x || (x == p.x && y > p.y);

	}

	friend bool operator== (const Point& p1,const Point& p2){

	return (p1.x==p2.x && p1.y==p2.y);

	}


	friend bool operator!= (const Point& p1,const Point& p2){

	return (!(p1.x==p2.x && p1.y==p2.y));

	}

};

llu Orientation(Point O, Point A, Point B)

{

	return (A.x - O.x) * (B.y - O.y)
```

```
            - (A.y - O.y) * (B.x - O.x);
}
Point p0;
Point NextToTop(vector<Point> &S)
{
      Point p = S.back();
      S.pop_back();
      Point res = S.back();
      S.push_back(p);
      return res;
}
double DistSq(Point p1, Point p2)
{
      return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
            (p1.y - p2.y)*(p1.y - p2.y));
}

int Compare(const void *vp1, const void *vp2)
{
      Point *p1 = (Point *)vp1;
      Point *p2 = (Point *)vp2;

      // Find orientation
      int o = Orientation(p0, *p1, *p2);
      if (o == 0) return DistSq(p0, *p1) - DistSq(p0, *p2);

      return o;
}

bool IsValueInVector(vector<Point> vec, Point value) {
```

```
    return find(vec.begin(), vec.end(), value) != vec.end();
}


int Tangent(vector<Point> v,Point p){
        int l = 0;
        int r = v.size();
        int l_before = Orientation(p, v[0], v[v.size()-1]);
        int l_after = Orientation(p, v[0], v[(l + 1) % v.size()]);
        while (l < r){
        int c = ((l + r)>>1);
        int c_before = Orientation(p, v[c], v[(c - 1) % v.size()]);
        int c_after = Orientation(p, v[c], v[(c + 1) % v.size()]);
        int c_side = Orientation(p, v[l], v[c]);
        if (c_before >=0 and c_after >=0)
                return c;
        else if ((c_side > 0) and (l_after < 0 or l_before == l_after) or (c_side < 0 and c_before
< 0))
                r = c;
        else
                l = c + 1 ;
        l_before = -c_after;
        l_after = Orientation(p, v[l], v[(l + 1) % v.size()]);
        }
        return l;
}
// Tim bao loi bang thuat toan Monotone Chain
vector<Point> MonotoneChainMethod (vector<Point> A, vector<Point>& M)
{
        int n = A.size(), k = 0;


        if (n <= 3)
```

```
            return A;
        vector<Point> ans(2 * n);
        // Build lower hull
        for (int i = 0; i < n; ++i) {
            while (k >= 2 && Orientation(ans[k - 2], ans[k - 1], A[i]) <= 0)
                k--;
            ans[k++] = A[i];
        }
        // Build upper hull
        for (size_t i = n - 1, t = k + 1; i > 0; --i) {
            while (k >= t && Orientation(ans[k - 2], ans[k - 1], A[i - 1]) <= 0)
                k--;
            ans[k++] = A[i - 1];
        }
        ans.resize(k - 1);
        for(int i = 0; i < n; i++){
       if(!IsValueInVector(ans, A[i])) M.push_back(A[i]);
    }
        return ans;
}
// Tim bao loi bang thuat toan Jarvis Wrapping
vector<Point> JarvisWrappingMethod (vector<Point> A)
{
        int n = A.size();
        vector<Point> ans;
        // Find the leftmost point
        int l = 0;
        int p = l, q;
        do
        {
```

```
                ans.push_back(A[p]);
                q = (p+1)%n;
                for (int i = 0; i < n; i++)
                {
                        if (Orientation(A[p], A[i], A[q]) < 0)
                                q = i;
                }
                p = q;
        } while (p != l);
        return ans;
}
// Tim bao loi bang thuat toan Graham Scan
vector<Point> GrahamScanMethod(vector<Point> A)
{
        int n = A.size();
        if(n <= 1) return A;
        // Find the bottommost point
        int ymin = A[0].y, min = 0;
        for (int i = 1; i < n; i++)
        {
                int y = A[i].y;

                if ((y < ymin) || (ymin == y && A[i].x < A[min].x))
                        ymin = A[i].y, min = i;
        }
        swap(A[0], A[min]);
        p0 = A[0];
        qsort(&A[1], n-1, sizeof(Point), Compare);
        int m = 1;
        for (int i=1; i<n; i++)
```

```
        {
                while (i < n-1 && Orientation(p0, A[i], A[i+1]) == 0)
                        i++;

                A[m] = A[i];
                m++;
        }
        vector<Point> ans;
        ans.push_back(A[0]);
        ans.push_back(A[1]);
        ans.push_back(A[2]);
        for (int i = 3; i < m; i++)
        {
                while (ans.size()>1 && Orientation(NextToTop(ans), ans.back(), A[i]) >= 0)
                        ans.pop_back();
                ans.push_back(A[i]);
        }
        return ans;
}
pair<int,int> Extreme_hullpt_pair(vector<vector<Point> >& hulls){
        int h = 0,p = 0;
        for (int i = 0; i < hulls.size(); ++i){
        int min_index = 0, min_y = hulls[i][0].y;
        for(int j = 1; j < hulls[i].size(); ++j){
                        if(hulls[i][j].y < min_y){
                min_y = hulls[i][j].y;
                min_index = j;
                        }
        }
        if(hulls[i][min_index].y < hulls[h][p].y){
```

```
                h = i;

                p = min_index;

        }

        }

        return make_pair(h,p);

}

pair<int,int> Next_hullpt_pair(vector<vector<Point> >& hulls, pair<int,int> lpoint){

        Point p = hulls[lpoint.first][lpoint.second];

        pair<int,int> next = make_pair(lpoint.first, (lpoint.second + 1) %
hulls[lpoint.first].size());

        for (int h=0; h< hulls.size(); h++){

        if(h != lpoint.first){

                int s= Tangent(hulls[h],p);

                Point q= hulls[next.first][next.second];

                Point r= hulls[h][s];

                int t= Orientation(p,q,r);

                if( t < 0 || (t == 0) && DistSq(p,r) > DistSq(p,q))

            next = make_pair(h,s);

        }

        }

        return next;

}

vector<Point> Keep_left (vector<Point>& v,Point p){

        while(v.size()>1 && Orientation(v[v.size()-2],v[v.size()-1],p) <= 0)

        v.pop_back();

        if(!v.size() || v[v.size()-1] != p)

        v.push_back(p);

        return v;

}

vector<Point> GrahamScan(vector<Point>& points){

        if(points.size()<=1)
```

```
        return points;
        qsort(&points[0], points.size(), sizeof(Point), Compare);
        vector<Point> lower_hull;
        for(int i=0; i<points.size(); ++i)
        lower_hull = Keep_left(lower_hull,points[i]);
        reverse(points.begin(),points.end());
        vector<Point> upper_hull;
        for(int i=0; i<points.size(); ++i)
        upper_hull = Keep_left(upper_hull,points[i]);
        for(int i=1;i<upper_hull.size();++i)
        lower_hull.push_back(upper_hull[i]);
        return lower_hull;
}
// Tim bao loi bang thuat toan Chan
vector<Point> ChansMethod(vector<Point> v){
        for(int t=0; t< v.size(); ++t){
        for(int m=1; m< (1<<(1<<t)); ++m){
                        vector<vector<Point> > hulls;
                        for(int i=0;i<v.size();i=i+m){
                vector<Point> chunk;
                if(v.begin()+i+m <= v.end())
                        chunk.assign(v.begin()+i,v.begin()+i+m);
                else
                                chunk.assign(v.begin()+i,v.end());
                hulls.push_back(GrahamScan(chunk));
                        }

                        vector<pair<int,int> > hull;
                        hull.push_back(Extreme_hullpt_pair(hulls));
                        for(int i=0; i<m; ++i){
```

```
                pair<int,int> p= Next_hullpt_pair(hulls,hull[hull.size()-1]);
                vector<Point> output;
                if(p==hull[0]){
                                for(int j=0; j<hull.size();++j){
                                        if (output.size() == 0 || output.front() !=
hulls[hull[j].first][hull[j].second])
                                output.push_back(hulls[hull[j].first][hull[j].second]);
                                }
                                return output;
                }
                hull.push_back(p);
                        }
        }
        }
}
// Tinh dien tich bao loi
double PolygonArea (vector<Point> M){
        double area = 0;
        int n = M.size();
        for (int i = 0; i < n; i++){
                area += M[i].x*M[(i+1)%n].y+M[(i+1)%n].x*M[i].y;
        }
        return area/2;
}
// Tim canh nho nhat cua bao loi
double MinDistOfConvexHull(vector<Point> M, vector<Point>& ans){
        int n = M.size();
        double min = DistSq(M[n-1], M[0]);
        int index = n-1;
        for (int i = 0; i < n-1; i++) {
                if (DistSq(M[i], M[i+1]) < min){
```

```
                min = DistSq(M[i], M[i+1]);
                index = i;
            }
        }
        ans.push_back(M[index]);
        ans.push_back(M[(index+1)%n]);

        return min;
}
int CompareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a,  *p2 = (Point *)b;
    return (p1->x != p2->x) ? (p1->x - p2->x) : (p1->y - p2->y);
}
int CompareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a,   *p2 = (Point *)b;
    return (p1->y != p2->y) ? (p1->y - p2->y) : (p1->x - p2->x);
}
double BruteForce(Point P[], int n)
{
    double min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (DistSq(P[i], P[j]) < min)
                min = DistSq(P[i], P[j]);
    return min;
}
double StripClosest(Point strip[], int size, double d, vector<Point>& ans)
{
```

```
    double min = d;
    int imin = 0;
    int jmin = 0;
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (DistSq(strip[i],strip[j]) < min) {
                    min = DistSq(strip[i], strip[j]);
                    imin = i;
                    jmin = j;
                            }
        ans.clear();
        ans.push_back(strip[imin]);
        ans.push_back(strip[jmin]);
    return min;
}
double ClosestUtil(Point Px[], Point Py[], int n, vector<Point>& ans)
{
    if (n <= 3)
        return BruteForce(Px, n);
    int mid = n/2;
    Point midPoint = Px[mid];
    Point Pyl[mid];
    Point Pyr[n-mid];
    int li = 0, ri = 0;
    for (int i = 0; i < n; i++)
    {
      if ((Py[i].x < midPoint.x || (Py[i].x == midPoint.x && Py[i].y < midPoint.y)) && li<mid)
        Pyl[li++] = Py[i];
      else
        Pyr[ri++] = Py[i];
```

```
    }
    double dl = ClosestUtil(Px, Pyl, mid, ans);
    double dr = ClosestUtil(Px + mid, Pyr, n-mid, ans);
    double d = min(dl, dr);
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(Py[i].x - midPoint.x) < d)
            strip[j] = Py[i], j++;
    return StripClosest(strip, j, d, ans);
}
double MinDistInside(vector<Point> X, vector<Point>& ans)
{
        int n = X.size();
        Point P[n];
        for (int i = 0; i < n; i++){
                P[i] = X[i];
        }
    Point Px[n];
    Point Py[n];
    for (int i = 0; i < n; i++)
    {
        Px[i] = P[i];
        Py[i] = P[i];
    }
    qsort(Px, n, sizeof(Point), CompareX);
    qsort(Py, n, sizeof(Point), CompareY);

    return ClosestUtil(Px, Py, n, ans);
}
```

```
float MinDistInside2(vector<Point> P, vector<Point>& ans)
{
        int size = P.size();
    float min = DistSq(P[0], P[size-1]);
        int imin = 0;
        int jmin = size - 1;
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size; ++j)
            if (DistSq(P[i], P[j]) < min){
                min = DistSq(P[i], P[j]);
                imin = i;
                jmin = j;
                    }
        ans.clear();
        ans.push_back(P[imin]);
        ans.push_back(P[jmin]);
    return min;
}
void PrintVector(vector<Point> A){
        for (int i = 0; i < A.size(); i++)
                cout << "(" << A[i].x << ", " << A[i].y << ")" << "; ";
}
bool compare_point(const Point& p1, const Point& p2) {
  return p1.x != p2.x || p1.y != p2.y;
}
int main()
{
        int n = (int)(rand()%10+10);
        vector<Point> A, M, B;
```

```
        set<Point, bool (*)(const Point&, const Point&)> input(compare_point);


  A.push_back({2, });
A.push_back({3, 7});
A.push_back({4, 3});
A.push_back({2, 9});
A.push_back({6, 12});
A.push_back({7, 16});
A.push_back({8, 3});
A.push_back({9, 8});
A.push_back({10, 7});
A.push_back({11, 12});

        // Sort points lexicographically
        sort(A.begin(), A.end());

        cout << "n = " << n << endl;
        cout << "Toa do cac diem la: \n";
        PrintVector(A);
        // Tim bao loi bang thuat toan Monotone Chain
        M = MonotoneChainMethod(A, B);
        cout << "\nToa do cac diem nam tren bao loi theo thuat toan Monotone Chain la: \n";
        PrintVector(M);

        cout << "\nToa do cac diem khong nam tren bao loi la: \n";
        PrintVector(B);

        // Tim bao loi bang thuat toan Jarvis Wrapping
        cout << "\nToa do cac diem nam tren bao loi theo thuat toan Jarvis Wrapping la: \n";
        PrintVector(JarvisWrappingMethod(A));
```

```
    // Tim bao loi bang thuat toan Graham Scan
    cout << "\nToa do cac diem nam tren bao loi theo thuat toan Graham Scan la: \n";
    PrintVector(GrahamScanMethod(A));

    // Tim bao loi bang thuat toan Chan
    cout << "\nToa do cac diem nam tren bao loi theo thuat toan Chan la: \n";
    PrintVector(ChansMethod(A));

    // Dien tich cua bao loi
    cout << "\n\nDien tich cua bao loi la: " << PolygonArea(M);

    return 0;
}
```

 # Trả lời: Dán kết quả thực thi vào bên dưới:

```
n = 13
Toa do cac diem la:
(2, 0); (2, 9); (3, 7); (4, 3); (6, 12); (7, 16); (8, 3); (9, 8); (10, 7); (11, 12);
Toa do cac diem nam tren bao loi theo thuat toan Monotone Chain la:
(2, 0); (8, 3); (10, 7); (11, 12); (7, 16); (2, 9);
Toa do cac diem khong nam tren bao loi la:
(3, 7); (4, 3); (6, 12); (9, 8);
Toa do cac diem nam tren bao loi theo thuat toan Jarvis Wrapping la:
(2, 0); (2, 9); (7, 16); (11, 12); (10, 7); (8, 3);
Toa do cac diem nam tren bao loi theo thuat toan Graham Scan la:
(2, 0); (2, 9); (7, 16); (11, 12); (10, 7); (8, 3);
Toa do cac diem nam tren bao loi theo thuat toan Chan la:
(2, 0); (8, 3); (10, 7); (11, 12); (7, 16); (2, 9);

Dien tich cua bao loi la: 331
```

Câu 4 (*2 điểm*): Cho hàm số $f(x) = \ln\left(2x^2+1\right)^2 + 9x + 3e^{x^2} - 5$.

a) *(1 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) γ, hệ số động lượng là α

 # Trả lời: dán sơ đồ khối hoặc mã giả

function gradientDescentWithMomentum(f_dh, x0, learning_rate, momentum_coefficient, max_iterations, epsilon):

```
    x = x0
    delta_x = -1

    for iteration in range(max_iterations):
        # Tính gradient tại điểm hiện tại
        gradient = f_dh(x)

        # Cập nhật delta_x theo phương pháp Momentum
        delta_x = delta_x * momentum_coefficient - learning_rate * gradient

        # Cập nhật vị trí mới của x
        x = x + delta_x

        # Kiểm tra điều kiện dừng
        if abs(gradient) < epsilon:
            break

    return x
```

b) *(1 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 ε:

 # Trả lời: Dán code vào bên dưới:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

const double GAMMA = 0.001;
const double alpha = 0.5;
const double epsilon = 0.00001;
const int MAX_LOOP = 1000;
```

```
// f_1(x) = (e^(2x) + x - 10)^2 + 2(x + 1)^2
double f_1(double x)
{
   // return pow((exp(2 * x) + x - 10), 2) + 2 * pow((x + 1), 2);
    return log(x+45)+2*exp(pow(x,5)-pow(x,3))-5*pow(x,3)-x+20;
}
// f_1_dh_1 = 6x + 4e^(2x)x + 4*e^(4x) - 38*e(2x) - 16
double f_1_dh_1(double x)
{
    //return 2 * (exp(2 * x) + x - 10) * (2 * exp(2 * x) + 1) + 4 * (x + 1);
        return 2*(5*pow(x,4)-3*pow(x,2))*exp(pow(x,5)-pow(x,3))+1/(x+45)-15*pow(x,2)-1;


}


double f_1_dh_2(double x)
{
  //  return 2 * pow(2 * exp(2 * x) + 1, 2) + 8 * (exp(2 * x) + x - 10) * exp(2 * x) + 4;
return 2 * pow(5 * pow(x, 4) - 3 * pow(x, 2), 2) * exp(pow(x, 5) - pow(x, 3))
           + 2 * (20 * pow(x, 3) - 6 * x) * exp(pow(x, 5) - pow(x, 3))
           - 1 / pow(x + 45.0, 2) - 30 * x;}


// f_2(x) = (e^(2x^2+1) - 2x + 1)^2 - 5(x + 1)^3



double gradientDescent(double (*f_dh_1)(double), double x0)
{
    double x = x0;
    for (int i = 0; i < MAX_LOOP; i++)
    {
        x -= GAMMA * f_dh_1(x);
```

```
        if (fabs(f_dh_1(x)) < epsilon)
        {
            break;
        }
    }

    return round(x * 100000) / 100000;
}
double gradientDescentWithMomentum(double (*f_dh_1)(double), double x0)
{
    double x = x0, x_temp;
    double delta_x = -1;
    for (int i = 0; i < MAX_LOOP; i++)
    {
        // x_temp = x; // luu lai gia tri truoc ne

        delta_x = delta_x * alpha - GAMMA * f_dh_1(x);

        x = x - GAMMA * f_dh_1(x) + delta_x * alpha;

        if (fabs(f_dh_1(x)) < epsilon)
        {
            break;
        }
    }

    return round(x * 100000) / 100000;
}
int main()
```

```
{
    double x0f_1 = 0.5;



    double xfgd = gradientDescent(f_1_dh_1, x0f_1);
    double xfgdm = gradientDescentWithMomentum(f_1_dh_1, x0f_1);
    cout << "f_1(x) =  ln(2x^2+1) ^2+9x +3*e^(x^2)-5 "
        << "\n"

        << "\tGD with Momentum:global min = " << f_1(xfgdm) << " at x = " << xfgdm <<
"\n";

}
```

 # Trả lời: **Dán kết quả thực thi** với điểm khởi $x = -1.00000$, tham số học học (*learning rate*) $\gamma = 0.001$, hệ số động lượng (*momentum coefficient*) là $\alpha = 0.5$, số bước lặp $N \geq 1000$ và sai số $\varepsilon = 10^{-5}$:

```
f_1(x) =  ln(2x^2+1) ^2+9x +3*e^(x^2)-5
        GD with Momentum:global min = 18.2464 at x = 1.18455
```

Câu 5 (*1 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:

$$P = \begin{bmatrix} 0 & 0.5 & 0.3 & 0.2 \\ 0.1 & 0.2 & 0.55 & 0.15 \\ 0.4 & 0.3 & 0.2 & 0.1 \\ 0 & 0.25 & 0.35 & 0.4 \end{bmatrix}$$

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 3. Tính xác suất hệ thống làm việc ở trạng thái 1 sau 1, 2 và 3 bước thời gian vận hành.

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

Đà Nẵng, ngày 04 tháng 12 năm 2023

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

KHOA CÔNG NGHỆ THÔNG TIN

(đã duyệt)