14th Marathon of Parallel Programming

SBAC-PAD & WSCAD – 2019

October 17th, 2019.

Rules for Local Contest

For all problems, read carefully the input and output session. For all problems, a sequential implementation is given, and it is against the output of those implementations that the output of your programs will be compared to decide if your implementation is correct. You can modify the program in any way you see fit, except when the problem description states otherwise. You must upload a compressed file (zip) with your source code, the *Makefile* and an execution script. The script must have the name of the problem. You can submit as many solutions to a problem as you want. Only the last submission will be considered. The *Makefile* must have the rule *all*, which will be used to compile your source code. The execution script runs your solution the way you design it – it will be inspected not to corrupt the target machine.

All *Local Teams* must use the computers that the organization provides. Only the judges have access to the judge machine.

The execution time of your program will be measured running it with *time* program and taking the real CPU time given. Each program will be executed at least three times with the same input and the mean time will be taken into account. The sequential program given will be measured the same way. You will earn points in each problem, corresponding to the division of the sequential time by the time of your program (*speedup*). The team with the most points at the end of the marathon will be declared the winner.

This problem set contains 6 problems; pages are numbered from 1 to 17.

General information

Compilation

You must use CC or CXX inside your *Makefile*. Be careful when redefining them! There is a simple *Makefile* inside you problem package that should be modified. Example:

```
FLAGS=-O3
EXEC=sum
CXX=icpc

all: $(EXEC)

$(EXEC):
	$(CXX) $(FLAGS) $(EXEC).cpp -c -o $(EXEC).o
	$(CXX) $(FLAGS) $(EXEC).o -o $(EXEC)
```

Running

You must have an execution script that has the same name of the problem. This script runs your solution the way you design it. There is a simple script inside you problem package that should be modified. Example:

```
$ cat A
#!/bin/bash
# This script runs Problem A
# export OMP_NUM_THREADS=32
# mpiexec –n 32 ./sum
./sum
```

Measure the execution time of your solution using *time* program. Add input/output redirection when collecting time. Use *diff* program to compare the original and your solution results. Example:

```
$ time –p ./A < original_input.txt > my_output.txt
real 4.94
user 0.08
sys 1.56

$ diff my_output.txt original_output.txt

$
```

Problem A

Maximum Sum Subsequence

Given an array sequence [A_1, A_2 …A_n], where n is the total amount of integer values (also the array size), this implementation aims to find the maximum possible sum of increasing subsequence *S* of length *k* such that $S_1 \leq S_2 \leq S_3 \leq S_4 \leq .... \leq S_k$.

Input

The input set contains only one test case. The first line contains one value: the array size (which is also the amount of elements to be read). The second line contains the length of the subsequence *S*, represented by *k*. Then, the last line contains a list of the elements to be inserted into the array (note that the number of elements must be equal to the array size).

The input must be read from the standard input.

Output

The output contains only one line printing the maximum possible sum of increasing subsequence *S*.

The output must be written to the standard output.

Example

Input example 1	Output example 1
8 3 8 5 9 10 5 6 21 8	40

```
int MaxIncreasingSub(int arr[], int n, int k)
{
    int **dp, ans = -1;
    dp = new int *[n];
    for(int i=0; i < n; i++)
        dp[i] = new int[k+1];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < k; j++){
            dp[i][j] = -1;
        }
    }
    for (int i = 0; i < n; i++) {
        dp[i][1] = arr[i];
    }
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (arr[j] < arr[i]) {
                for (int l = 1; l <= k - 1; l++) {
                    if (dp[j][l] != -1) {
                        dp[i][l + 1] =
max(dp[i][l + 1],dp[j][l] + arr[i]);
                    }
                }
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if (ans < dp[i][k])
            ans = dp[i][k];
    }
    return (ans == -1) ? 0 : ans;
}
```

Problem B

Brute-Force Password Cracking

One way to crack a password is through a brute-force algorithm, which tests all possible combinations of a password exhaustively until it finds the correct one.

A common practice in security systems is to store the *hash* of user passwords (instead of the plan text). Hashes map a given x (the password) with variable length N to a given $y = hash(x)$ with a fixed length M, following specific security properties. One of them is: you cannot obtain the value of x from the value of y. In a security system, y is stored instead of x and the comparison is performed directly between the hash of attempted password x' and the stored $y = hash(x)$.

Examples of hash algorithms are MD5 and SHA. For MD5 algorithm, the fixed length of the $hash(x)$ is 128 bits usually presented as 32-hexadecimal characters.

This problem considers the cracking of MD5-hashed password by trying exhaustively all the combinations of possible characters (uppercase and lower-case letters and numeric symbols) and comparing pairs of MD5 hashes. That is, for all combinations of characters x'_i, $hashMD5(x) = hashMD5(x'_i)$?

Input

An input consists of only one case of test. The single line contains a string with 32-hexadecimal characters representing the value of MD5 hash of the password to be cracked.

Consider that the possible passwords used to generate the hashes have the length N, with $1 \leq N \leq 10$.

The input must be read from the standard input.

Output

The output is a single line that contains the password value found.

The output must be written to the standard output.

Example

Input example 1	Output example 1
`7a95bf926a0333f57705aeac07a362a2`	`found: PASS`

```c
void print_digest(byte * hash){
	int x;

	for(x = 0; x < MD5_DIGEST_LENGTH; x++)
		printf("%02x", hash[x]);
	printf("\n");
}

/*
 * This procedure generate all combinations of possible
letters
*/
void iterate(byte * hash1, byte * hash2, char *str, int idx,
int len, int *ok) {
	int c;

	// 'ok' determines when the algorithm matches.
	if(*ok) return;
	if (idx < (len - 1)) {
		// Iterate for all letter combination.
		for (c = 0; c < strlen(letters) && *ok==0; ++c)
{
			str[idx] = letters[c];
			// Recursive call
			iterate(hash1, hash2, str, idx + 1, len,
ok);
		}
	} else {
		// Include all last letters and compare the
hashes.
		for (c = 0; c < strlen(letters) && *ok==0; ++c)
{
			str[idx] = letters[c];
			MD5((byte *) str, strlen(str), hash2);
			if(strncmp((char*)hash1, (char*)hash2,
MD5_DIGEST_LENGTH) == 0){
				printf("found: %s\n", str);
				*ok = 1;
			}
		}
	}
}

/*
 * Convert hexadecimal string to hash byte.
*/
void strHex_to_byte(char * str, byte * hash){
	char * pos = str;
	int i;

	for (i = 0; i < MD5_DIGEST_LENGTH/sizeof *hash; i++) {
		sscanf(pos, "%2hhx", &hash[i]);
		pos += 2;
	}
}
```

Problem F

Closest Pair of Points

Given an array of n points in the 2D plane, the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For instance, it can be used to monitor airplanes that come too close together in an air-traffic control system, thus avoiding possible collisions. The following formula is used to determine the distance between two points *p* and *q* in a 2D plane:

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

The given sequential program calculates the distance between the closest pair of points in $O(n(\log\ n)^2)$ time using a Divide and Conquer strategy. Your task is to improve the performance of the program using parallel strategies.

Input

The input contains only one test case. The first line of input contains an integer *N* ($2 \leq N \leq 80{,}000{,}000$), which denotes the number of points. Each of the next *N* lines contains two floats *X* and *Y* ($-10{,}000{,}000 \leq X, Y \leq 10{,}000{,}000$ with exactly 3 digits after the decimal), which denote the coordinates of the i-th point. There is no coincident point in the input data.

The input must be read from the standard input.

Output

The output must contain a single line with a float *D*, denoting the distance between the closest pair of points. *D* must contain exactly 6 digits after the decimal.

The output must be written to the standard output.

Example

Input example 1	Output example 1
5 -5.000 2.000 -15.000 -7.000 0.000 0.000 6.000 3.000 2.000 4.000	4.123106

```
inline double abs2(double a) {
    if (a < 0.0)
        return -a;
    return a;
}

bool compX(const Point& a, const Point& b) {
    if (abs2(a.x-b.x)<EPS)
        return a.y<b.y;
    return a.x<b.x;
}

bool compY(const Point& a, const Point& b) {
    if (abs2(a.y-b.y)<EPS)
        return a.x<b.x;
    return a.y<b.y;
}

inline double sqr(double a) {
    return a * a;
}

inline double distance(Point a, Point b) {
    return sqr(a.x - b.x) + sqr(a.y - b.y);
}

double solve(int l, int r) {
    double minDist = INF;
    double dist;
    int i, j;
    if(r-l+1 <= BRUTEFORCESIZE){
        for(i=l; i<=r; i++){
            for(j = i+1; j<=r; j++) {
                dist = distance(point[i], point[j]);
                if(dist<minDist){
                    minDist = dist;
                }
            }
        }
        return minDist;
    }

    int m = (l+r)/2;
    double dL = solve(l,m);
    double dR = solve(m,r);
    minDist = (dL < dR ? dL : dR);

    int k = l;
    for(i=m-1; i>=l && abs(point[i].x-point[m].x)<minDist; i--){
        border[k++] = point[i];
    }
    for(i=m+1; i<=r && abs(point[i].x-point[m].x)<minDist; i++){
        border[k++] = point[i];
    }

    if (k-l <= 1) return minDist;

    sort(&border[l], &border[l]+(k-l), compY);

    for(i=l; i<k; i++){
        for(j=i+1; j<k && border[j].y - border[i].y < minDist; j++){
            dist = distance(border[i], border[j]);
            if (dist < minDist){
                minDist = dist;
            }
        }
    }

    return minDist;
}
```