21BDS0340

Abhinav Dinesh Srivatsa

Design and Analysis of Algorithms Lab

Assignment 5

Question 1

Code:

```
#include <iostream>
#include <regex>
#include <math.h>
using namespace std;

template <typename T>
class Stack
{
public:
    T *arr;
    int top;
    int length;

    Stack<T>(int n)
    {
        this->length = n;
        this->arr = new T[n];
        top = -1;
    }

    bool isEmpty()
    {
        return top == -1;
    }

    bool isFull()
    {
        return top == length - 1;
    }

    void push(T n)
    {
        if (!isFull())
            this->arr[++top] = n;
    }

    T pop()
    {
        if (!isEmpty())
```

```
            return this->arr[top--];
        return this->arr[0];
    }
};

int main()
{
    string input;
    getline(cin, input);
    regex exp("-*[0-9]+");
    smatch res;
    int digit;
    Stack<int> *digits = new Stack<int>(input.length());
    while (regex_search(input, res, exp, regex_constants::match_any))
    {
        digits->push(stoi(res[0]));
        input = res.suffix().str();
    }

    int n = pow(digits->top + 1, 0.5);
    int **matrix = new int *[n];
    for (int x = 0; x < n; x++)
        matrix[x] = new int[n];
    for (int x = n - 1; x >= 0; x--)
        for (int y = n - 1; y >= 0; y--)
        {
            digit = digits->pop();
            if (digit != -1)
                matrix[x][y] = digit;
            else
                matrix[x][y] = 10000;
        }

    for (int x = 0; x < n; x++)
        for (int i = 0; i < n; i++)
            if (i != x)
                for (int j = 0; j < n; j++)
                    if (j != x)
                        matrix[i][j] = matrix[i][j] < matrix[i][x] + matrix[x][j] ?
matrix[i][j] : matrix[i][x] + matrix[x][j];

    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            if (matrix[x][y] == 10000)
                matrix[x][y] = -1;

    for (int x = 0; x < n; x++)
    {
        for (int y = 0; y < n; y++)
            cout << matrix[x][y] << " ";
        cout << "\n";
```

```
    }

    free(digits);
    free(matrix);
}
```

Input:
{{0, 16, 13 ,-1, -1, -1}, {-1, 0, 10, 12, -1,- 1}, {-1, 4, 0, -1, 14, -1}, {-1, -1, 9, 0, -1, 20}, {-1, -1, -1, 7, 0, 4}, {-1, -1, -1, -1, -1, 0}}

Output: (-1 represents infinity)
0 16 13 28 27 17
-1 0 10 12 24 1
-1 4 0 16 14 5
-1 13 9 0 23 14
-1 20 16 7 0 4
-1 -1 -1 -1 -1 0

Question 2

Code:

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

struct Point
{
    int x, y;
};

struct Segment
{
    Point left, right;
};

struct Event
{
    int x, y;
    bool isLeft;
    int index;
    Event(int x, int y, bool l, int i) : x(x), y(y), isLeft(l), index(i) {}

    bool operator<(const Event &e) const
    {
        if (y == e.y)
            return x < e.x;
        return y < e.y;
    }
};

bool onSegment(Point p, Point q, Point r)
```

```
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;

    return false;
}

int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;

    return (val > 0) ? 1 : 2;
}

bool doIntersect(Segment s1, Segment s2)
{
    Point p1 = s1.left, q1 = s1.right, p2 = s2.left, q2 = s2.right;

    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    if (o1 == 0 && onSegment(p1, p2, q1))
        return true;

    if (o2 == 0 && onSegment(p1, q2, q1))
        return true;

    if (o3 == 0 && onSegment(p2, p1, q2))
        return true;

    if (o4 == 0 && onSegment(p2, q1, q2))
        return true;

    return false;
}

set<Event>::iterator pred(set<Event> &s, set<Event>::iterator it)
{
    return it == s.begin() ? s.end() : --it;
}
```

```
set<Event>::iterator succ(set<Event> &s, set<Event>::iterator it)
{
    return ++it;
}

int isIntersect(Segment arr[], int n)
{
    unordered_map<string, int> mp;
    vector<Event> e;
    for (int i = 0; i < n; ++i)
    {
        e.push_back(Event(arr[i].left.x, arr[i].left.y, true, i));
        e.push_back(Event(arr[i].right.x, arr[i].right.y, false, i));
    }

    sort(e.begin(), e.end(), [](Event &e1, Event &e2)
         { return e1.x < e2.x; });

    set<Event> s;
    int ans = 0;
    for (int i = 0; i < 2 * n; i++)
    {
        Event curr = e[i];
        int index = curr.index;

        if (curr.isLeft)
        {
            auto next = s.lower_bound(curr);
            auto prev = pred(s, next);
            bool flag = false;
            if (next != s.end() && doIntersect(arr[next->index], arr[index]))
            {
                string s = to_string(next->index + 1) + " " + to_string(index + 1);
                if (mp.count(s) == 0)
                {
                    mp[s]++;
                    ans++;
                }
            }
            if (prev != s.end() && doIntersect(arr[prev->index], arr[index]))
            {
                string s = to_string(prev->index + 1) + " " + to_string(index + 1);
                if (mp.count(s) == 0)
                {
                    mp[s]++;
                    ans++;
                }
            }
            if (prev != s.end() && next != s.end() && next->index == prev->index)
                ans--;
```

```
            s.insert(curr);
        }

        else
        {
            auto it = s.find(Event(arr[index].left.x, arr[index].left.y, true,
index));
            auto next = succ(s, it);
            auto prev = pred(s, it);

            if (next != s.end() && prev != s.end())
            {
                string s = to_string(next->index + 1) + " " + to_string(prev->index
+ 1);
                string s1 = to_string(prev->index + 1) + " " + to_string(next-
>index + 1);
                if (mp.count(s) == 0 && mp.count(s1) == 0 && doIntersect(arr[prev-
>index], arr[next->index]))
                    ans++;
                mp[s]++;
            }

            s.erase(it);
        }
    }

    for (auto &pr : mp)
    {
        cout << "Line: " << pr.first << "\n";
    }
    return ans;
}

int main()
{
    Segment arr[] = {{{1, 5}, {4, 5}}, {{2, 5}, {10, 1}}, {{3, 2}, {10, 3}}, {{6,
4}, {9, 4}}, {{7, 1}, {8, 1}}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of intersection points: " << isIntersect(arr, n);
    return 0;
}
```

Input:
{{{1, 1}, {10, 1}}, {{1, 2}, {10, 2}}, {-5, -5}, {0, 0}, {1, 1}, {10, 10}}

Output:
Intersections:
{{1, 1}, {10, 1}} and {{1, 1}, {10, 10}}
{{1, 2}, {10, 2}} and {{1, 1}, {10, 10}}

Question 3

Code:

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

struct Point
{
    int x, y;
};

struct Segment
{
    Point left, right;
};

struct Event
{
    int x, y;
    bool isLeft;
    int index;
    Event(int x, int y, bool l, int i) : x(x), y(y), isLeft(l), index(i) {}

    bool operator<(const Event &e) const
    {
        if (y == e.y)
            return x < e.x;
        return y < e.y;
    }
};

bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;

    return false;
}

int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;

    return (val > 0) ? 1 : 2;
}
```

```
bool doIntersect(Segment s1, Segment s2)
{
    Point p1 = s1.left, q1 = s1.right, p2 = s2.left, q2 = s2.right;

    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    if (o1 == 0 && onSegment(p1, p2, q1))
        return true;

    if (o2 == 0 && onSegment(p1, q2, q1))
        return true;

    if (o3 == 0 && onSegment(p2, p1, q2))
        return true;

    if (o4 == 0 && onSegment(p2, q1, q2))
        return true;

    return false;
}

set<Event>::iterator pred(set<Event> &s, set<Event>::iterator it)
{
    return it == s.begin() ? s.end() : --it;
}

set<Event>::iterator succ(set<Event> &s, set<Event>::iterator it)
{
    return ++it;
}

int isIntersect(Segment arr[], int n)
{
    unordered_map<string, int> mp;
    vector<Event> e;
    for (int i = 0; i < n; ++i)
    {
        e.push_back(Event(arr[i].left.x, arr[i].left.y, true, i));
        e.push_back(Event(arr[i].right.x, arr[i].right.y, false, i));
    }

    sort(e.begin(), e.end(), [](Event &e1, Event &e2)
         { return e1.x < e2.x; });

    set<Event> s;
```

```
    int ans = 0;
    for (int i = 0; i < 2 * n; i++)
    {
        Event curr = e[i];
        int index = curr.index;

        if (curr.isLeft)
        {
            auto next = s.lower_bound(curr);
            auto prev = pred(s, next);
            bool flag = false;
            if (next != s.end() && doIntersect(arr[next->index], arr[index]))
            {
                string s = to_string(next->index + 1) + " " + to_string(index + 1);
                if (mp.count(s) == 0)
                {
                    mp[s]++;
                    ans++;
                }
            }
            if (prev != s.end() && doIntersect(arr[prev->index], arr[index]))
            {
                string s = to_string(prev->index + 1) + " " + to_string(index + 1);
                if (mp.count(s) == 0)
                {
                    mp[s]++;
                    ans++;
                }
            }
            if (prev != s.end() && next != s.end() && next->index == prev->index)
                ans--;

            s.insert(curr);
        }

        else
        {
            auto it = s.find(Event(arr[index].left.x, arr[index].left.y, true,
index));
            auto next = succ(s, it);
            auto prev = pred(s, it);

            if (next != s.end() && prev != s.end())
            {
                string s = to_string(next->index + 1) + " " + to_string(prev->index
+ 1);
                string s1 = to_string(prev->index + 1) + " " + to_string(next-
>index + 1);
                if (mp.count(s) == 0 && mp.count(s1) == 0 && doIntersect(arr[prev-
>index], arr[next->index]))
                    ans++;
```

```
                mp[s]++;
            }

            s.erase(it);
        }
    }

    for (auto &pr : mp)
    {
        cout << "Line: " << pr.first << "\n";
    }
    return ans;
}

int main()
{
    Segment arr[] = {{{1, 5}, {4, 5}}, {{2, 5}, {10, 1}}, {{3, 2}, {10, 3}}, {{6,
4}, {9, 4}}, {{7, 1}, {8, 1}}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of intersection points: " << isIntersect(arr, n);
    return 0;
}
```

Input:
{{{1, 5}, {4, 5}}, {{2, 5}, {10, 1}}, {{3, 2}, {10, 3}}, {{6, 4}, {9, 4}}, {{7, 1}, {8, 1}}}

Output:
Intersections:
{{2, 5}, {10, 1}} and {{3, 2}, {10, 3}}