Competitive Programming

From Problem 2 Solution in O(1)

Computational Geometry

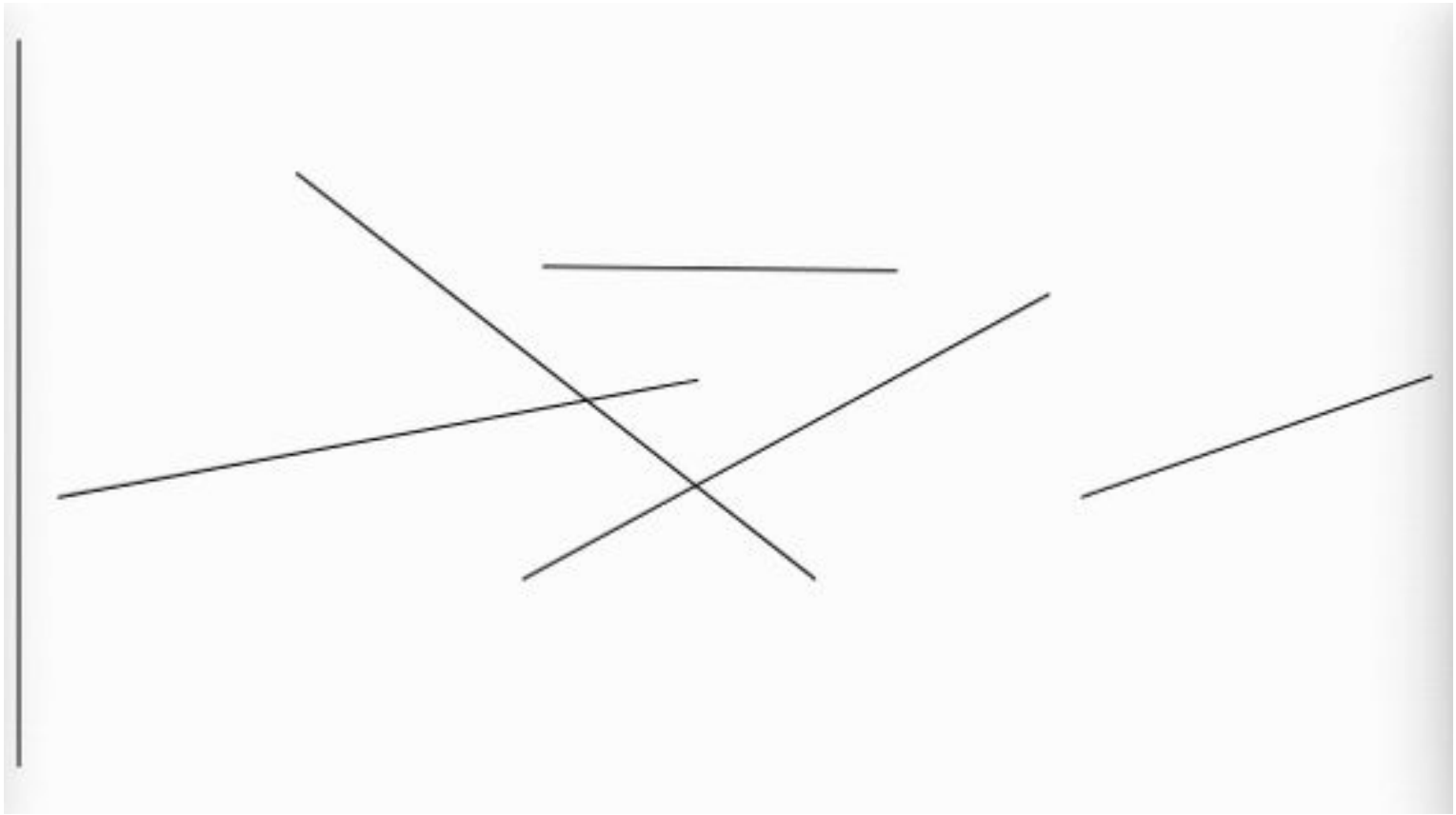
Line Sweep - Segments Intersection

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Line Sweep

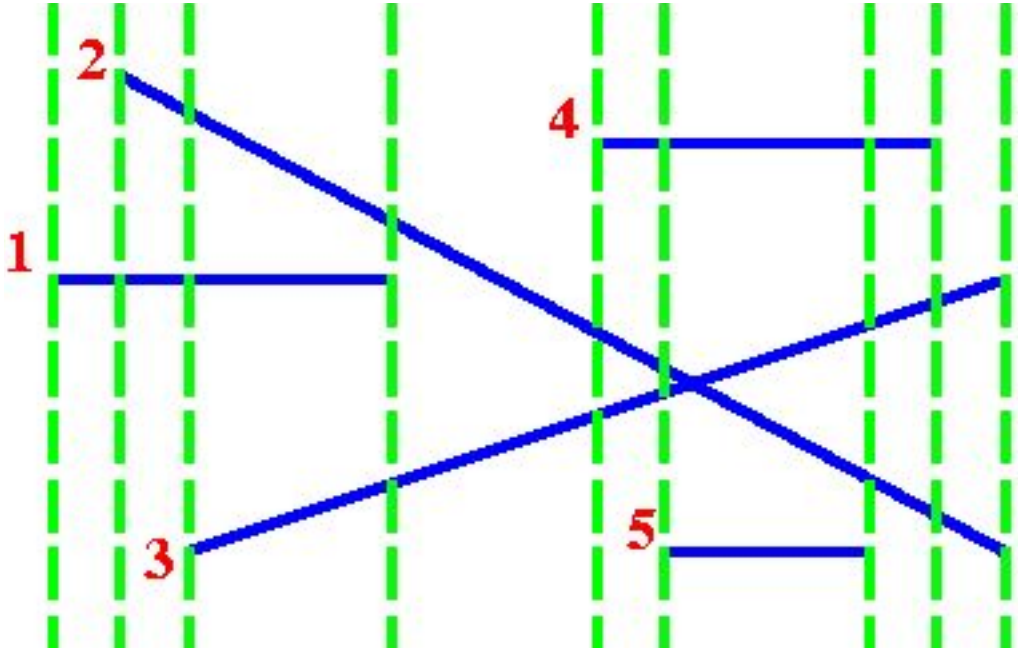
- Giving a plane of objects and some task
  - E.g. Set of segments to intersect
  - E.g. Set of rectangles to compute intersection
- One can consider every pair of objects!
  - However, every object (e.g. segment) interacts (e.g. intersect) with some surrounding ones **NOT** ALL
- Imagine a **vertical line** that is **swept** across the plane, specially **at discrete points** (events)
  - E.g. start/end of segment or a rectangle
  - We will use them to identify the **surrounding objects**

Line Sweep



Src: http://cglab.ca/~morin/teaching/2402/notes/planesweep.pdf

Line Sweep (using vertical line)

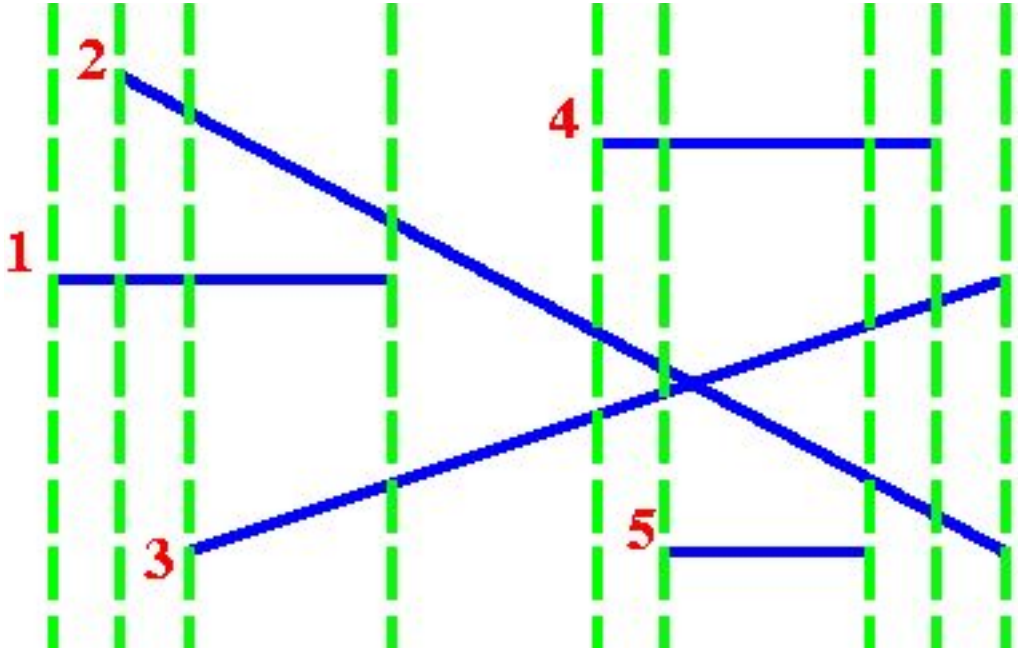


- Given N segments, find their **intersections**
- Naive solution is O(n^2): intersect every pair

- 5 segments (blue)
- 10 end points
- 10 sweep **events** (vertical lines)
- Each endpoint is **event**
- Event: start / end of segment

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)



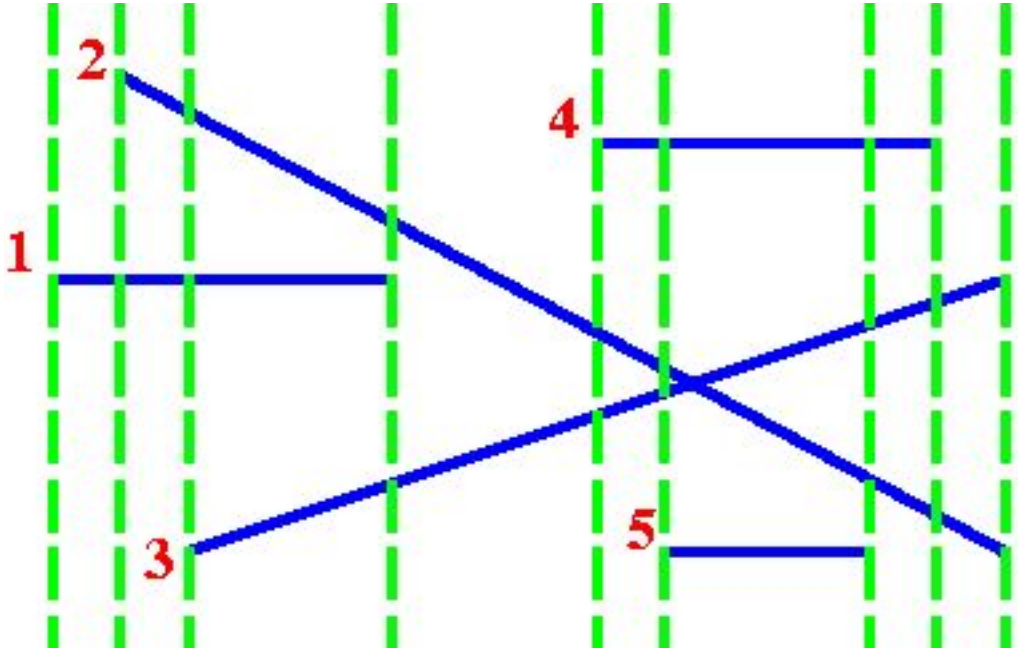
- Instead of considering ALL other segments
- We need to identify related subset only
- Let's call them the **active set**

- Whenever event occurs
  - It only consider the active set
  - it may use subset of it or all of it
  - If start event = add to active set
  - If end event = remove from active set
  - Processing of start event might be different from end event

- Let 4S (start of segment 4 event)
- Let 4E (end of segment 4 event)
- And so on for other segments

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)



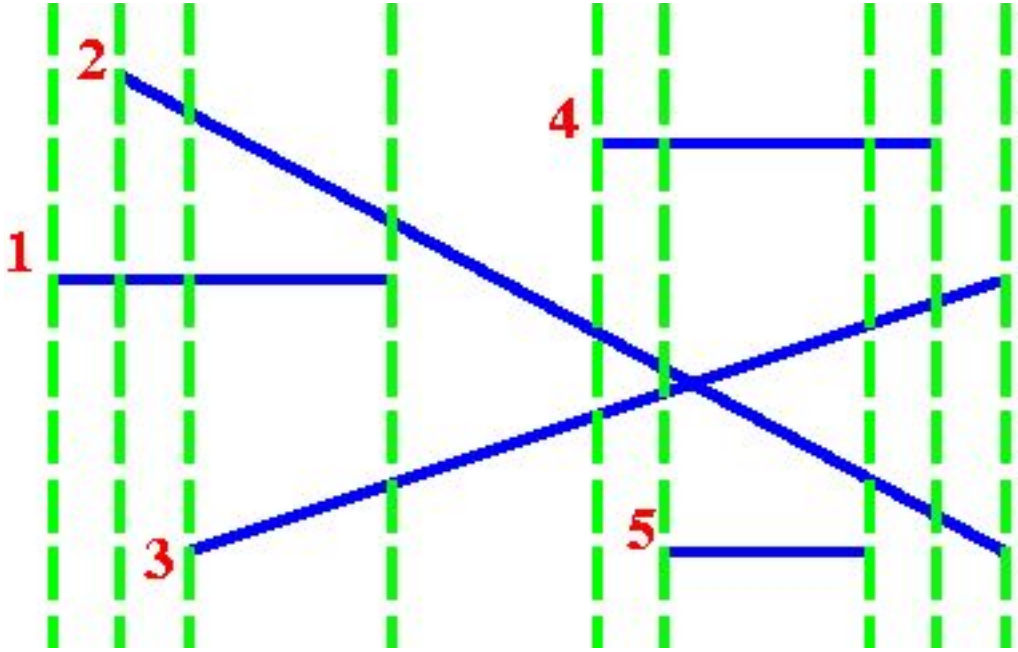
- Let 4S (start of segment 4 event)
- Let 4E (end of segment 4 event)
- And so on for other segments

- Given that we seep vertically
- Then we can think of sorting segments based on X-axis

- **Events sorting**:
- 1S, 2S, 3S, 1E, 4S, 5S, 5E, 4E, 3E, 2E

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

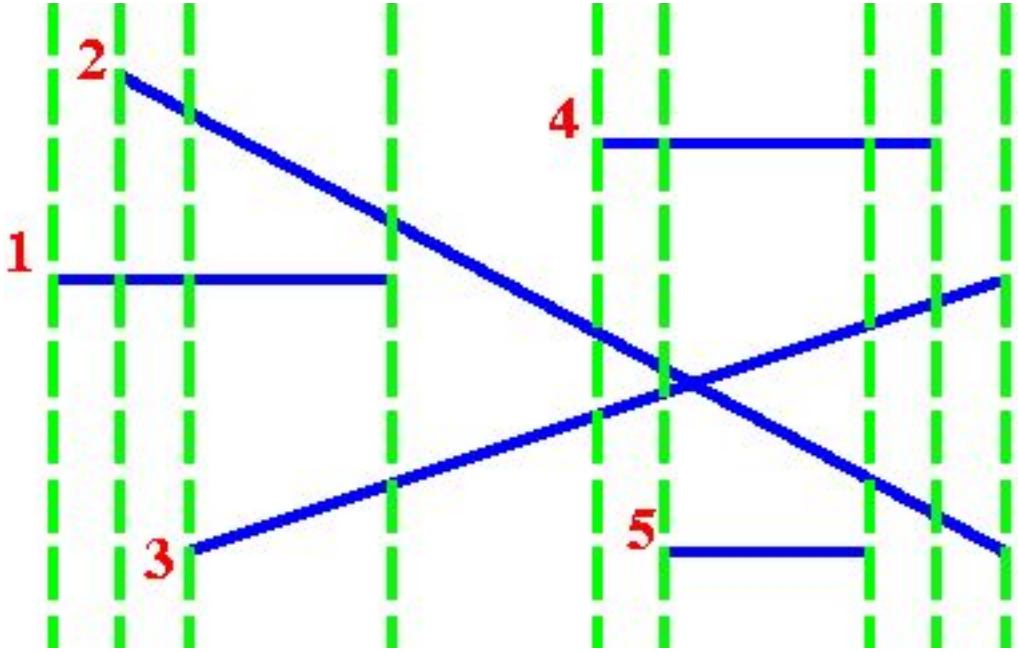
Line Sweep (using vertical line)



- **1S, 2S, 3S, 1E, 4S, 5S, 5E, 4E, 3E, 2E**
- What might be **active set** for event **4E**?
  - Ignore all event after it
  - Remove events that started/ended
- **1S, 2S, 3S, 1E, 4S, 5S, 5E, 4E, 3E, 2E**
- **1S, 2S, 3S, 1E, 4S, 5S, 5E, 4E**
- **, 2S, 3S, , 4S, , , 4E**
- So only segments {2, 3} are active for 4E
- Look at 4E in the figure **vertically**

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

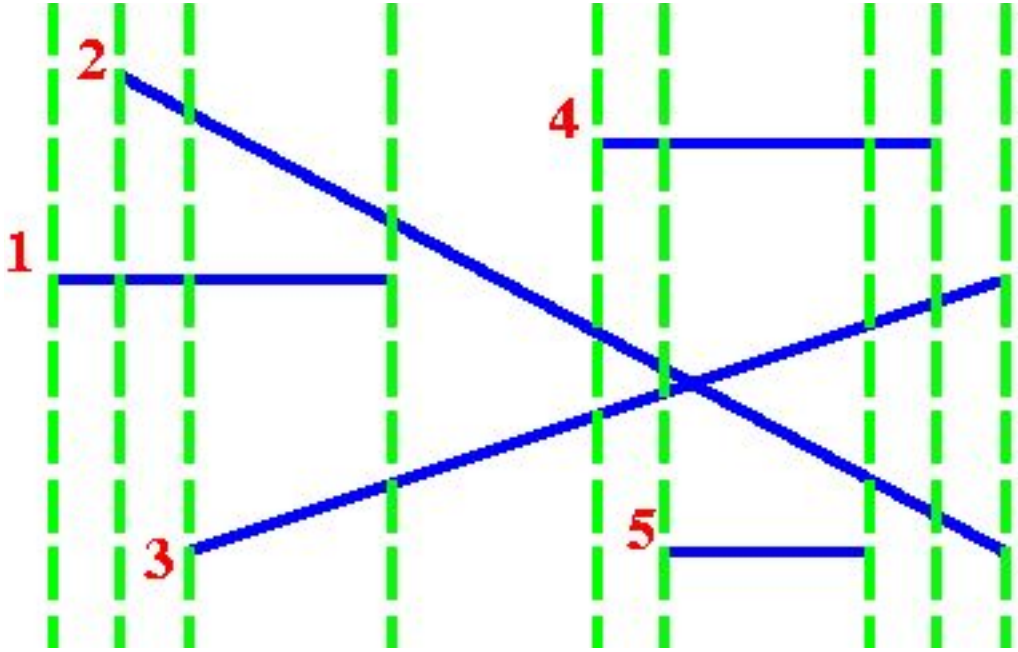
Line Sweep (using vertical line)



- **1S, 2S, 3S, 1E, 4S, 5S, 5E, 4E, 3E, 2E**
- What might be **active set** for event **1S**?
  - Segments {} ⇒ Empty
- What might be **active set** for event **2S**?
  - Segments {1}
- What might be **active set** for event **1E**?
  - Segments {2, 3} => Not same as 1S
- What might be **active set** for event **5S**?
  - Segments {2, 3, 4}
- What might be **active set** for event **3E**?
  - Segments {2}
- What might be **active set** for event **2E**?
  - Segments {} ⇒ Empty

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)



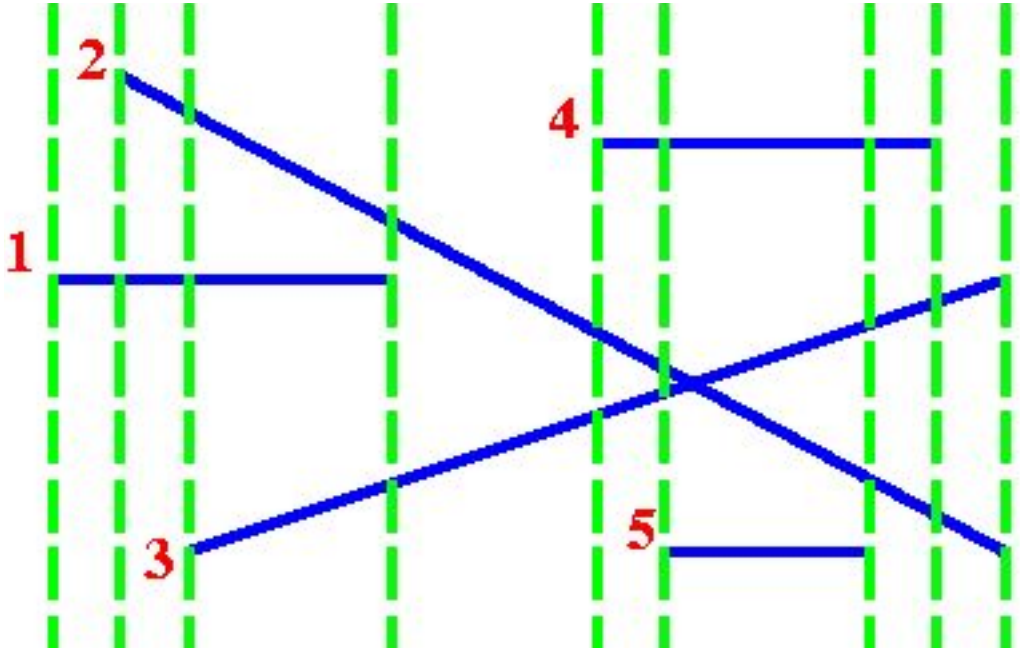
- **active set** for event **5S**? Segments {2, 3, 4}
- **active set** for event **5E**? Segments {2, 3, 4}

- **Questions** to think about?!
- Q1) Should we check intersections
  - at start event only?
  - end event? Or both?
- Q2) When we try to intersect: Should we try with **all** active set elements or a **subset**?
- Q3) What to check at each event? Is it **same** checkings for both endpoints?

Finding the right questions & answers is the major concern in Line Sweep Algorithms

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)



Bently-Ottman Algorithm

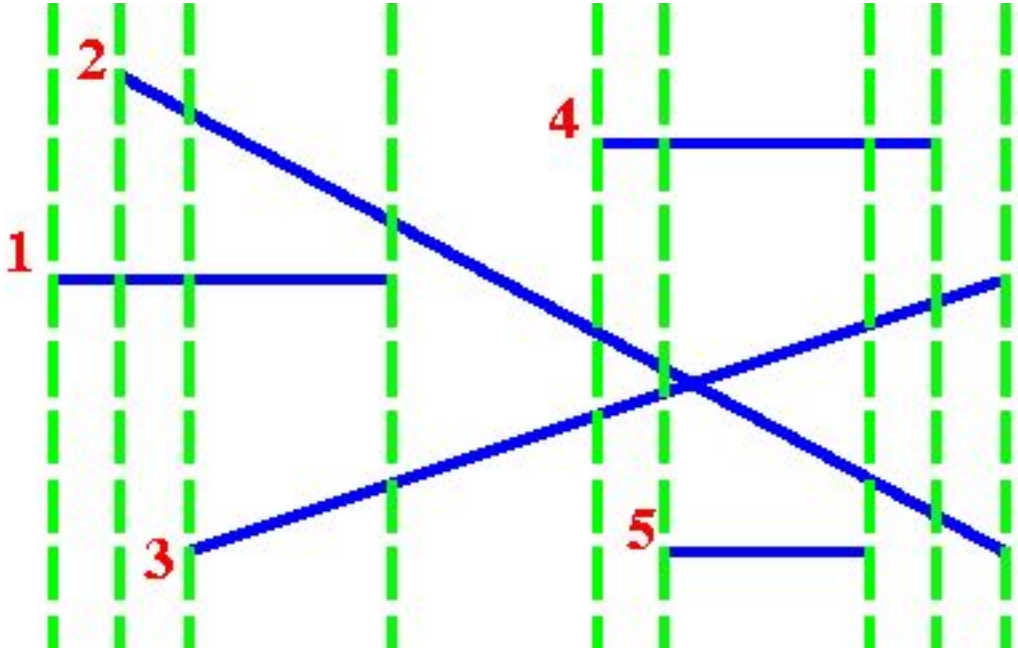
- Let current segment is C
- From active set, let segment above C is A and segment below C is B

- **For Start Event**
  - Check intersect(Current, Above)
  - Check intersect(Current, Below)
  - Add C to active Set

- **For End Event**
  - Remove C from active Set
  - Check intersect(Above, Below)

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

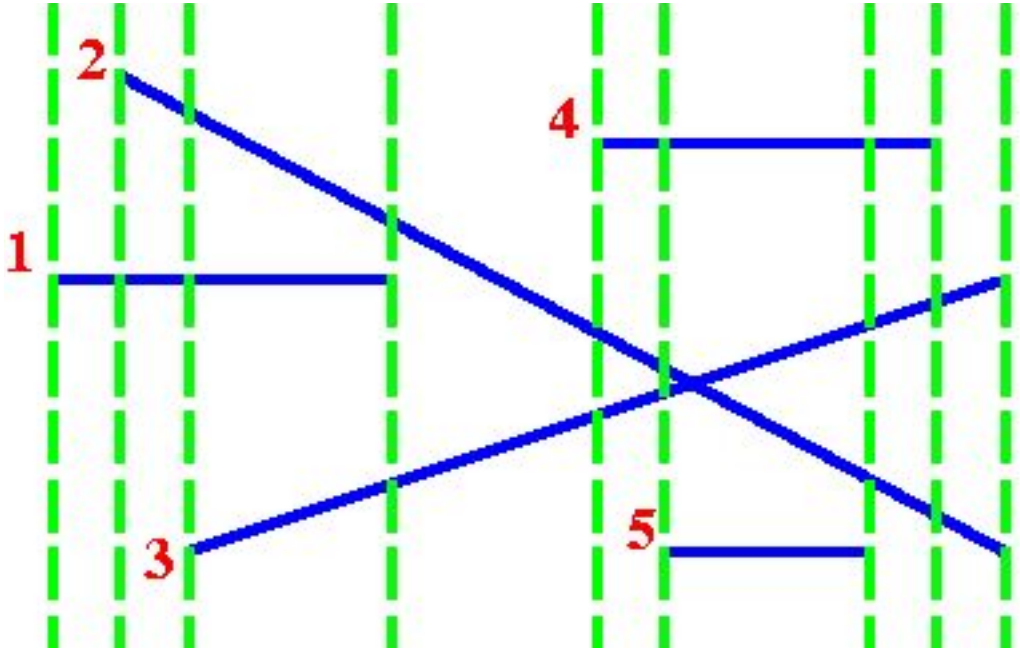


Events Processing

- Active List = {}
- Current Event **1S**
  - above = {}
  - below = {}
  - No intersection tests
- Add 1 to active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

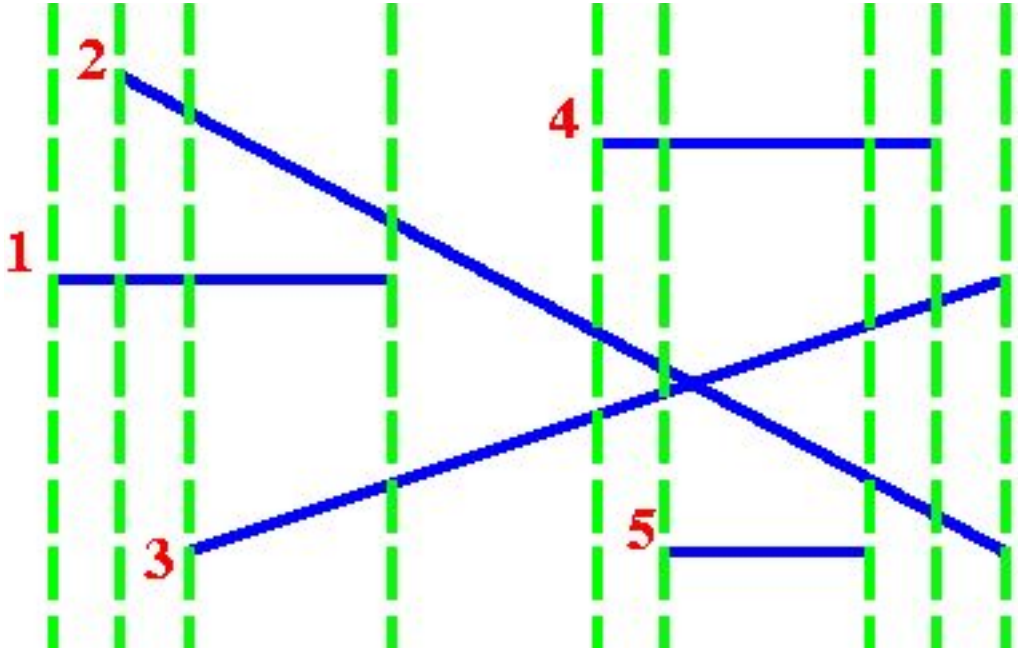


Events Processing

- Active List = {1}
- Current Event **2S**
  - above = {}
  - below = {1}
  - intersection(2, 1) = False
- Add 2 to active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

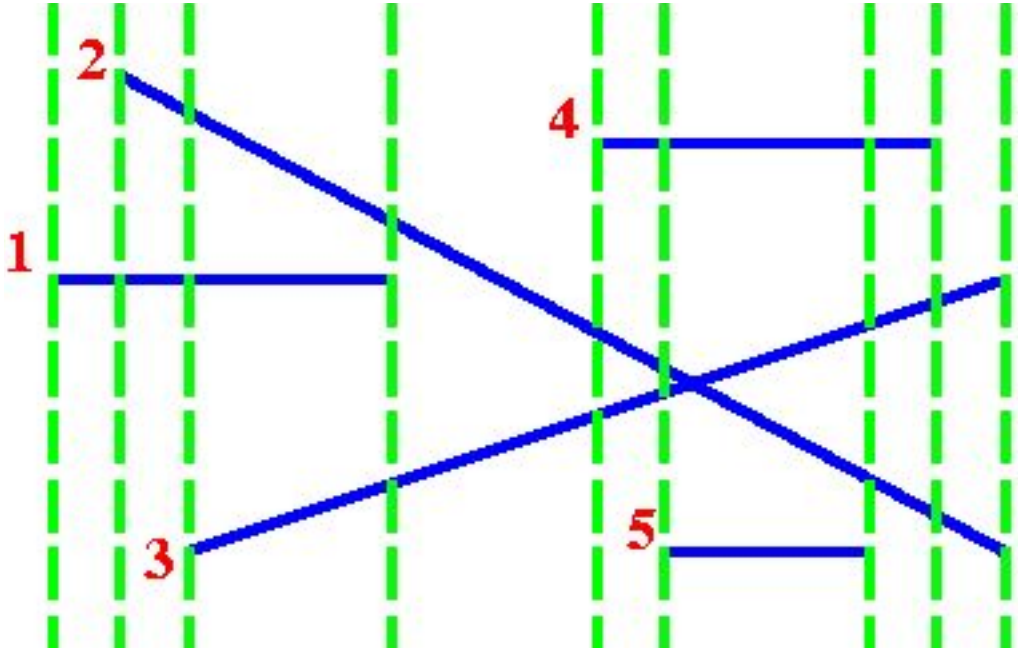


Events Processing

- Active List = {1, 2}
- Current Event **3S**
  - above = {1}
  - below = {}
  - intersection(3, 1) = False
- Add 3 to active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

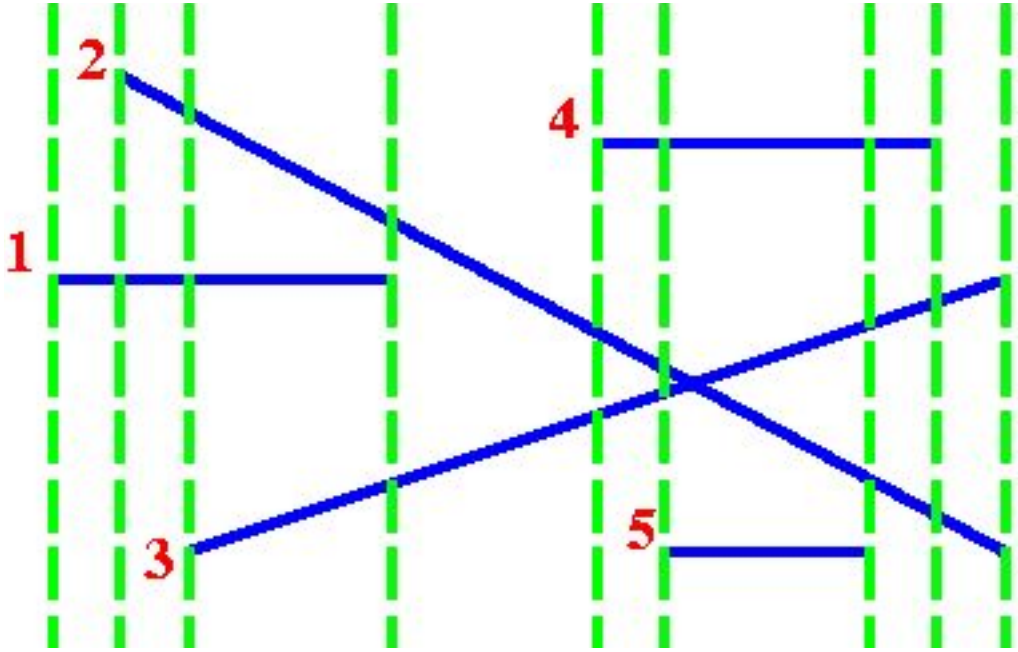


Events Processing

- Active List = {1, 2, 3}
- Current Event **1E**
  - above = {2}
  - below = {3}
  - intersections(2, 3) = True
- **Remove** 1 from active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

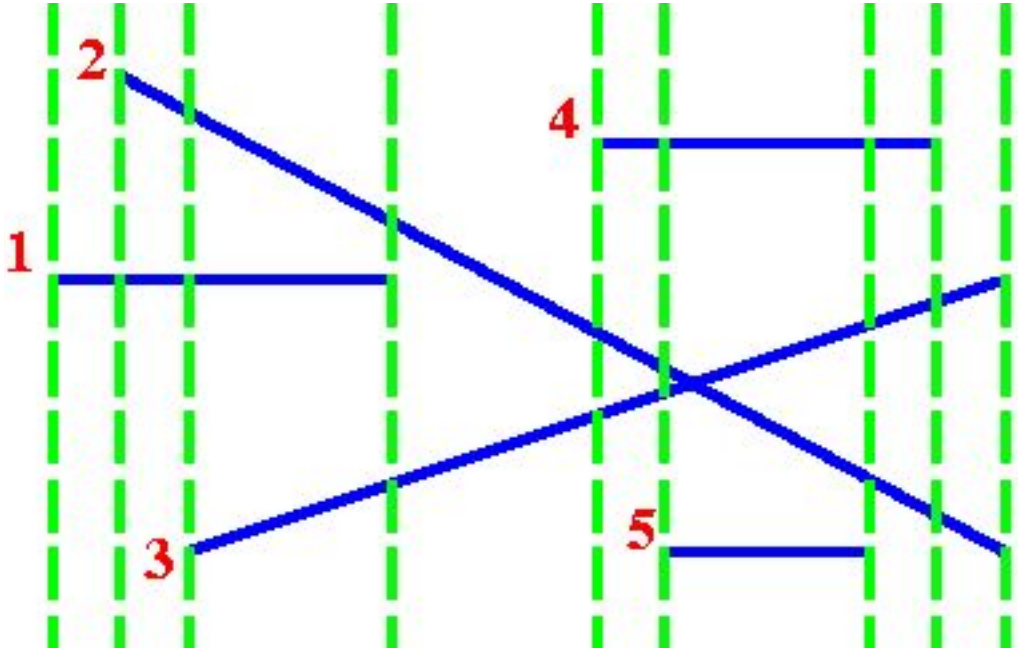


Events Processing

- Active List = {2, 3}
- Current Event **S4**
  - above = {2}
  - below = {3}
  - intersection(4, 2) = False
  - intersection(4, 3) = False
- Add 4 to active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

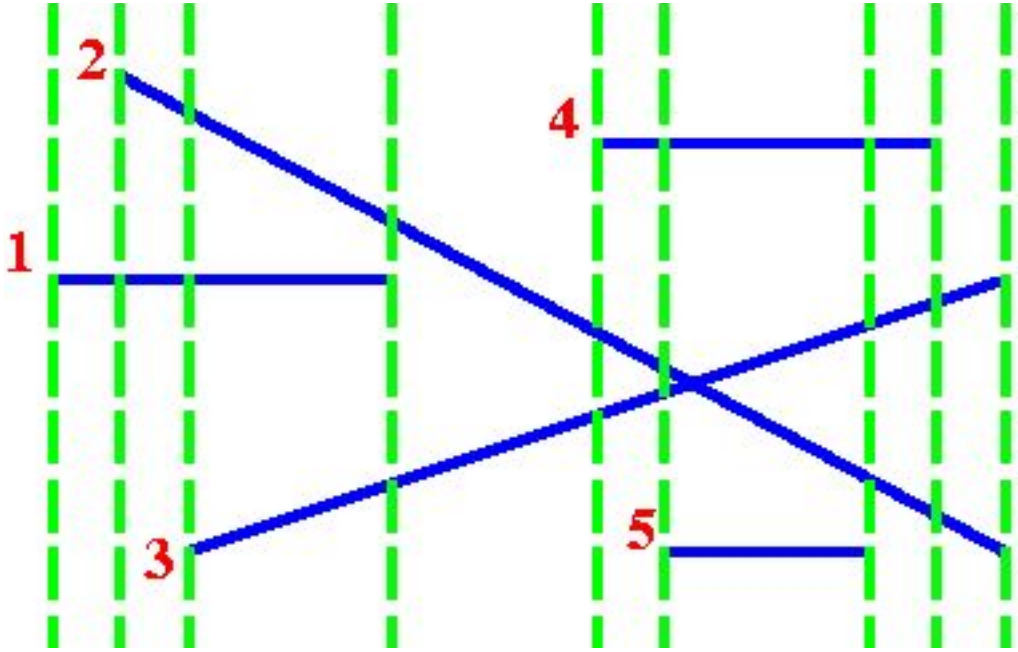


Events Processing

- Active List = {2, 3, 4}
- Current Event **S5**
  - above = {3}
  - below = {}
  - intersection(5, 3) = False
- Add 5 to active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

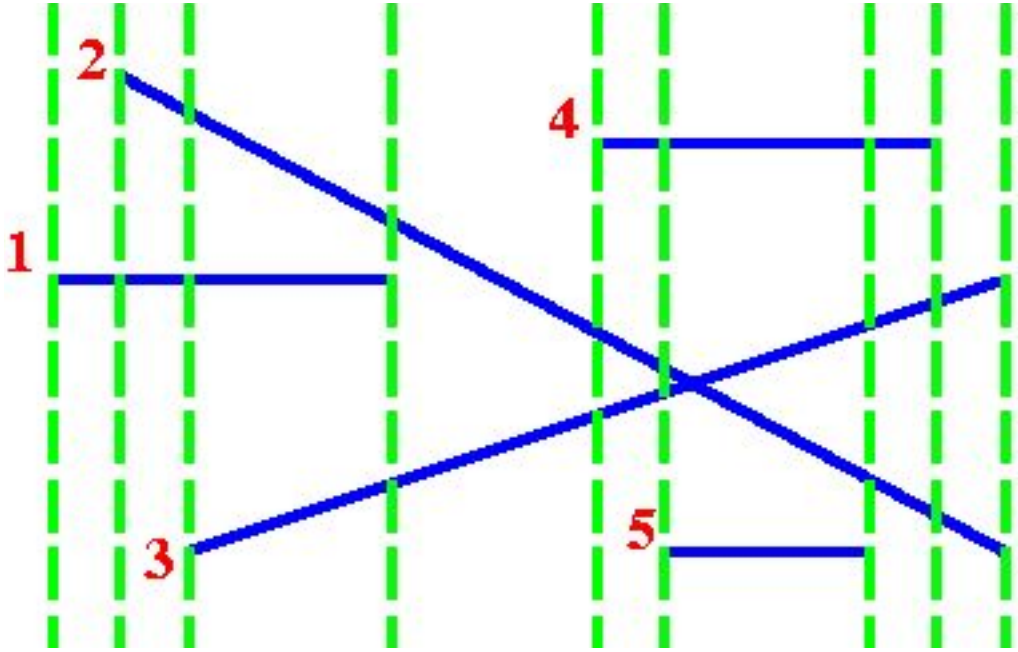


Events Processing

- Active List = {2, 3, 4, 5}
- Current Event **E5**
  - above = {3}
  - below = {}
  - No intersection tests
- Remove 5 from active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

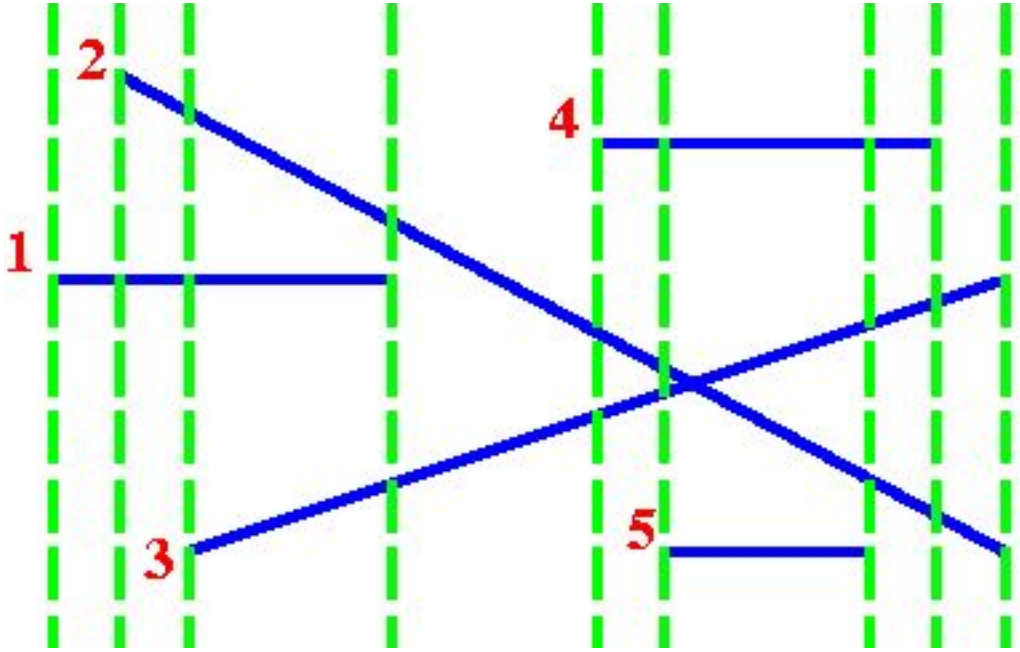


Events Processing

- Active List = {2, 3, 4}
- Current Event **E4**
  - above = {2}
  - below = {3}
  - intersections(2, 3) = YES
- Remove 4 from active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

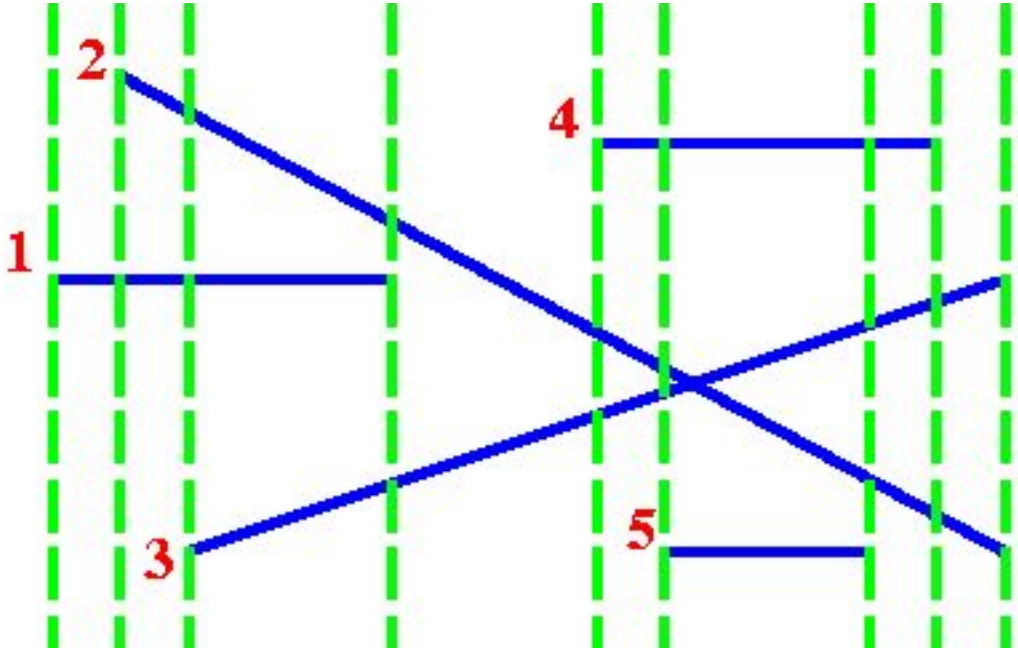


Events Processing

- Active List = {2, 3}
- Current Event **E3**
  - above = {2}
  - below = {}
  - No intersection tests
- Remove 3 from active list
- Similarly, remove 2 from active list

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Line Sweep (using vertical line)

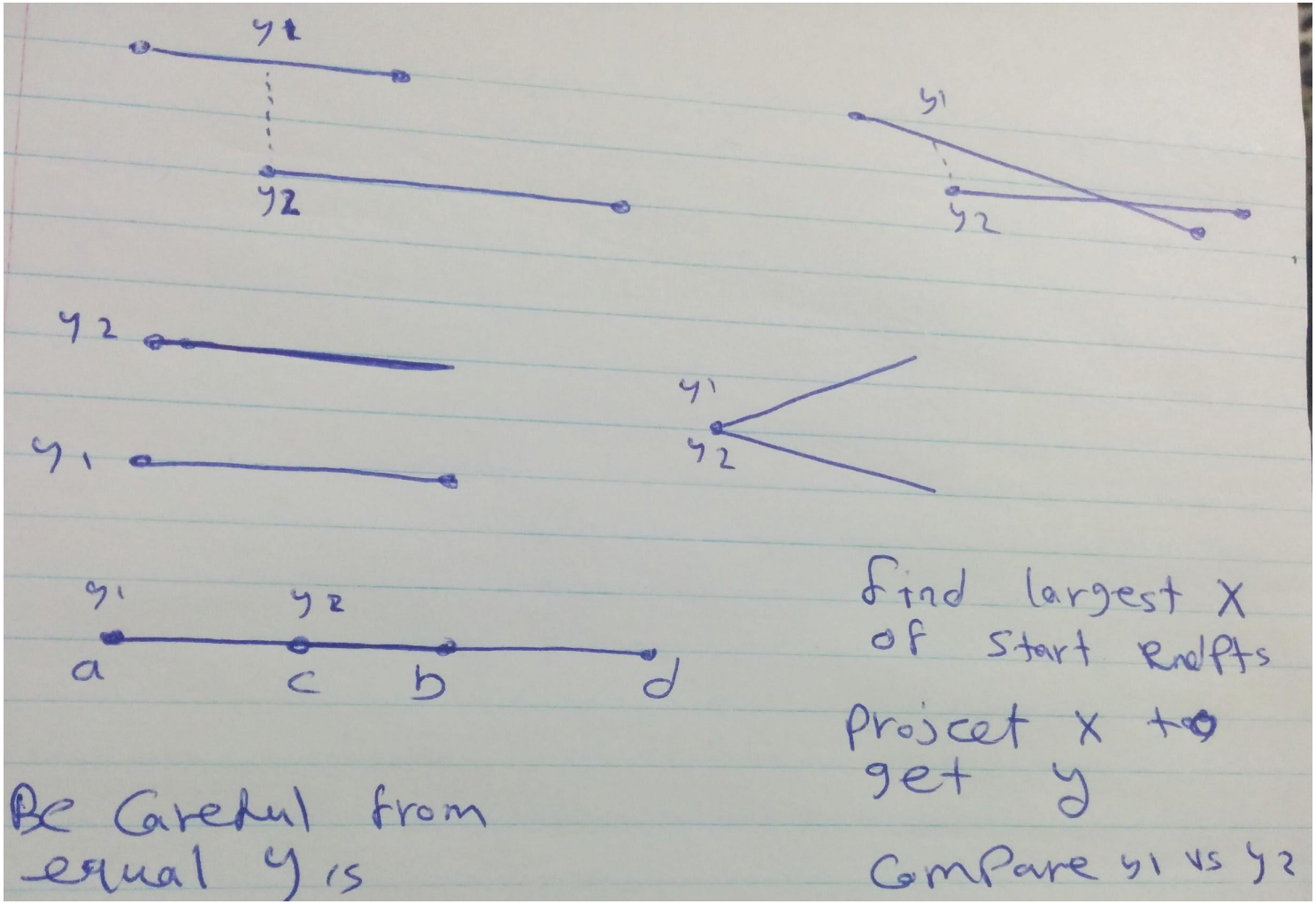


Notes

- Any intersecting pairs will eventually **become adjacent**...hence detected
- The **variability** of checks at start and end event surrounds the different cases
- E.g. During S3, 3 couldn't check intersect with above (2), because 1 is barrier in-between.
- However, when 1E occurs, it check above and below, which found intersection
- In other words, if at both start & end we check current segment against above/blow we will miss this case.

Src: http://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/

Is segment A below B?



Data Structures

- Segment Structure
  - 2 points of a segment
  - sorting segments based on how is lower (in y)
- Insert/Find/Delete Segments Structure
  - To do that efficiently, one needs Balanced Tree (e.g. AVL)
  - One can use C++ set/Priority Queue most of time
  - However, in some problem, you must use your Tree
- Events structure
  - endpoint point, event type, parent segment
  - sorting events based on X, **if tie ENTRY events first**

Algorithm Flow

- Read Segments...Create & Sort Events
- Iterate on events
- If entry event
  - Find BTree find below & above segments of current
  - intersect(current, below)? intersect(current, above)?
  - Insert it.
- If exist event
  - Find BTree find below & above segments of current
  - intersect(above, below)?
  - Delete it.

Segment Struct

```
struct segment {
  point p, q;
  int seg_idx;

  segment() {seg_idx = -1;}
  segment(point p_, point q_, int seg_idx_) {
    if (q_ < p_)
      swap(p_, q_);
    p = p_, q = q_, seg_idx = seg_idx_;
  }

  double CY(int x) const {
    if (dcmp(p.X, q.X) == 0)
      return p.Y; // horizontal

    double t = 1.0 * (x - p.X)/(q.X - p.X);
    return p.Y + (q.Y - p.Y)*t;
  }
  // operator< is very tricky and can cause 100 WAs.
  bool operator<(const segment& rhs) const {
    if(same(p, rhs.p) && same(q, rhs.q))
      return false;

    int maxX = max(p.X, rhs.p.X);
    int yc = dcmp(CY(maxX), rhs.CY(maxX));

    if (yc == 0) // critical condition
      return seg_idx < rhs.seg_idx;
    return yc < 0;
  }
};
```

Event Structure

```
int ENTRY = +1, EXIT = -1;          // entry types
const int MAX_SEGMENTS = 50000 + 9;
const int MAX_EVENTS = MAX_SEGMENTS * 2;

struct event {
  point p;
  int type, seg_idx;
  // smaller X first. If tie: ENTRY event first. Last on smaller Y
  bool operator <(const event & rhs) const {
    if (dcmp(p.X, rhs.p.X) != 0)
      return dcmp(p.X, rhs.p.X) < 0;
    if (type != rhs.type)
      return type > rhs.type;
    return dcmp(p.Y, rhs.p.Y) < 0;
  }
};

int n;
segment segments[MAX_SEGMENTS];
event events[MAX_EVENTS];
set&lt;segment&gt; sweepSet;
typedef set&lt;segment&gt;::iterator ITER;
```

Read Segments..Create/Order Events

```
cin >> n;
lp(i, n)
{
  cin >> x >> y;        point p1 = point(x, y);
  cin >> x >> y;        point p2 = point(x, y);
  segments[i] = segment(p1, p2, i);
}
bentleyOttmann_lineSweep();
```

```
void bentleyOttmann_lineSweep() {   // O( (k+n) logn )
  // Prepare events
  lp(i, n)
  {
    events[2*i] = {segments[i].p, ENTRY, i};
    events[2*i+1] = {segments[i].q, EXIT, i};
  }
  sort(events, events+2*n);
```

Sweeping over the events

```
lp(i, 2*n) {
  if (events[i].type == ENTRY) {
    auto status = sweepSet.insert(segments[events[i].seg_idx]);
    ITER cur = status.first, below = before(cur), above = after(cur);

    if(!status.second) {
      FoundIntersection(cur->seg_idx, events[i].seg_idx); // Duplicate
    } else {
      if(intersectSeg(cur, above))
        FoundIntersection(cur->seg_idx, above->seg_idx);
      if(intersectSeg(cur, below))
        FoundIntersection(cur->seg_idx, below->seg_idx);
    }
  } else {
    ITER cur = sweepSet.find(segments[events[i].seg_idx]);

    if(cur == sweepSet.end())
      continue; // e.g. Duplicate

    ITER below = before(cur), above = after(cur);

    if(intersectSeg(above, below))
      FoundIntersection(above->seg_idx, below->seg_idx);
    sweepSet.erase(cur);
  }
}
```

Little Utilities

```
bool intersectSeg(ITER seg1Iter, ITER seg2Iter) {
  if (seg1Iter == sweepSet.end() || seg2Iter == sweepSet.end())
    return false;
  return intersect(seg1Iter->p, seg1Iter->q, seg2Iter->p, seg2Iter->q);
}

ITER after(ITER cur) {
  return cur == sweepSet.end() ? sweepSet.end() : ++cur;
}

ITER before(ITER cur) {
  return cur == sweepSet.begin() ? sweepSet.end() : --cur;
}

void FoundIntersection(int i, int j) {
  printf("%d %d\n", i + 1, j + 1);
}
```

Simple Apps

- **Test if polygon is simple**
  - Recall that, a polygon is simple if it doesn't have intersecting segments
  - Note, consecutive polygon edges intersect at endpoint
  - We implemented before O(n^2) solution
  - One can do trivial adjustment to this code to check if any 2 edges intersect or not
- **Test if two polygons intersect**
  - Again, direct test if any of edges intersect
  - Again, special care for endpoints

Harder Apps

- Intersection (or union, or difference) of two simple polygons
  - Determine all intersection points, and use them as new vertices to construct the answer
- Polygon Triangulation
  - See1, See2

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً