Line Sweep Algorithms

Schalk-Willem Krüger – 2009 Training Camp 1

presentation.start();
Closest Pair Algorithm

The problem:

2008 Training Camp 2: Good Neighbours

The problem:

- Bruce wants to visit his friends on weekends.
- The friends are scattered around (each at a unique location).
- Find the two friends that live closest to each other.
- Maximum of 1 000 000 friends

Brute force: $O(N^2)$ – too slow
Initialize $h$ (shortest distance found so far) with the distance between the first two points. $h = \text{Euclidian distance between point 1 and 2} = 6.23 \text{ units}$
Closest Pair Algorithm

\[ h = \text{Euclidian distance between point 1 and 2} = 6.23 \text{ units} \]

No distance less than 6.23 units.
$h = \text{Euclidian distance between point 1 and 2} = 6.23 \text{ units}$

There are two points that are closer than the current value of $h$! Change $h$ to 5.45.
Closest Pair Algorithm

$h = \text{Euclidian distance between point 1 and 2} = 5.45 \text{ units}$

Change $h$ to 4.30 units.
Closest Pair Algorithm

- C++ implementation (with STL)

1. #include <stdio.h>
2. #include <set>
3. #include <algorithm>
4. #include <cmath>
5. using namespace std;
6. #define px second
7. #define py first
8. typedef pair<long long, long long> pairll;
9. int n;
10. pairll pnts [100000];
11. set<pairll> box;
12. double best;
13. int compx(pairll a, pairll b) { return a.px<b.px; }
14. int main () {
15.     scanf("%d", &n);
16.     for (int i=0;i<n;++i) scanf("%lld %lld", &pnts[i].px, &pnts[i].py);
17.     sort(pnts, pnts+n, compx);
18.     best = 1500000000; // INF
19.     box.insert(pnts[0]);
20.     int left = 0;
21.     for (int i=1;i<n;++i) {
22.         while (left<i && pnts[i].px-pnts[left].px > best) box.erase(pnts[left++]);
23.         for (typeof(box.begin()) it=box.lower_bound(make_pair(pnts[i].py-best, pnts[i].px-best));
24.             it! =box.end() && pnts[i].py+best>=it->py; it++)
25.             best = min(best, sqrt(pow(pnts[i].py - it->py, 2.0)+pow(pnts[i].px - it->px, 2.0)));
26.         box.insert(pnts[i]);
27.     }
28.     printf("%.2f\n", best);
29.     return 0;
30.}

Time complexity: O(N log N)
Closest Pair Algorithm

- C++ implementation (with STL) – zoomed in

19. `box.insert(pnts[0]);`
20. `int left = 0;`
21. `for (int i=1;i<n;++i) {
    while (left<i && pnts[i].px-pnts[left].px > best)
        box.erase(pnts[left++]);
    for (typeof(box.begin()) it=
        box.lower_bound(make_pair(pnts[i].py-best, pnts[i].px-best));
        it!=box.end() && pnts[i].py+best>=it->py; it++)
        best = min(best, sqrt(pow(pnts[i].py – it->py, 2.0)+
            pow(pnts[i].px - it->px, 2.0)));
    box.insert(pnts[i]);
} 24. `}
25. `printf("%.2f\n", best);`

Time complexity: O(N log N)
Line segment intersections (HV)

- Problem: given a set $S$ of $N$ closed segments in the plane, report all intersection points among the segment in $S$
- First consider the problem with only horizontal and vertical line segments
- Brute force: $O(N^2)$ time
  – too slow
Line segment intersections (HV)

- Use events: start of horizontal line, end of horizontal line and vertical line.
- Set contains all horizontal lines cut by the sweep line (sorted by Y). Indicated as red lines on diagram.
- Horizontal line event: add/remove line from set.
- Vertical line event: get all horizontal lines it cuts (get range from set). Indicated as red dots on diagram.
- Use balanced binary tree (C++ set) – guarantee $O(\log N)$ for operations.
Line segment intersections (HV)

1. // <Headers, structs, declarations, etc.>
2. // type=0: Starting point of horizontal line
3. // type=1: Ending point of horizontal line
4. int main () {
5.   // <Input>
6.   sort(events, events+e); // Sort events by X-coordinate
7.   for (int i=0;i<e;++i) {
8.     event c = events[i]; // c: current event
9.     if (c.type==0) s.insert(c.p1); // Add starting point to set
10.    else if (c.type==1) s.erase(c.p2); // Remove ending point from set
11.    else {
12.      for (typeof(s.begin()) it=s.lower_bound(point(-1, c.p1.y));
13.        it!=s.end() && it->y<=c.p2.y; it++) // Range search
14.        printf("%d, %d\n", events[i].p1.x, it->y);
15.      }
16.    }
17.   return 0;
18. }

Line segment intersections

- More general case: lines don't have to be horizontal or vertical.
- Lines change places when they intersect.
- Use priority queue to handle events.
- Events also sorted by X.
- Events in priority queue:
  - end-points of line-segments
  - intersection points of adjacent elements.
- Set contains segments that are currently intersecting with the sweep line.
Line segment intersections

- At a starting point of a line segment:
  - Insert segment into set.
  - Neighbours are no longer adjacent. Delete their intersection point (if any) from the priority queue if it exists.
  - Compute intersection of this point and its neighbours (if any) and insert into priority queue.

- At an ending point of a line segment:
  - Delete segment from set.
  - Neighbours are now adjacent. Compute their intersection point (if any) and insert into priority queue.

- At an intersection point of two line segments:
  - Output point.
  - Swap position of intersection segments in set.
  - The swapped segments have new neighbours now. Insert / delete intersecting points from priority queue (if needed).
Area of union of rectangles

- Calculate area of the union of a set of rectangles.
- Again work with events (sorted by x) and a set (sorted by y).
- Events:
  - Left edge
  - Right edge
- Set contains all the rectangles the sweep line is crossing.
Area of union of rectangles

- Know x-distance ($\Delta x$) between two adjacent events.
- Multiply it by the cut length of the sweep line ($\Delta y$) to get the total area of rectangles between the two events.
- Do this by running the same algorithm rotated 90 degrees. (Horizontal sweep line running from top to bottom)
  - Use only rectangles in the active set
  - Event: Horizontal edges.
  - Use a counter that indicates how many rectangles are overlapping at the current point.
  - Cut lengths are between two events where the count is zero.
Area of union of rectangles

Example of inner loop

$\Delta x$ between two adjacent vertical events

Horizontal sweep line

Vertical sweep line
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Example of inner loop

Count: 3
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Example of inner loop
Area of union of rectangles

Source code (C++) can be seen on the handout.
Area of union of rectangles

- Run time: $O(N^2)$ with boolean array in the place of a balanced binary tree (pre-sort the set of horizontal edges).
- Can be reduced to $O(N \log N)$ with a binary tree manipulation.
Given: A number of rectangles. (0<=N<5000)

Calculate the perimeter (length of the boundary of the union of all rectangles)

Use basically the same algorithm

Horizontal boundaries: where count is zero in inner loop.
Convex hull with sweep line

- Graham scan: Sort by angle – is expensive and can get numeric errors. (Can use C++ complex library)
- Simpler algorithm: Andrew's Algorithm
- Sort by X and use a sweep line!
- Upper hull: Start at point with minimum X coordinate and move right. When the last three points aren't convex, delete the second-last point. Repeat until the last three points form a convex triangle.
- Sweep line algorithm runs in \( O(N) \)
- \( O(N \log N) \) – (points are sorted)
Questions?

presentation.end();