Making Barriers Is Fun

Writer : platypus

問題概要

- 二次元平面状に点がN個ある。
- 以下の二つの条件どちらかを満たす線分を平面に追加していく。
  既存の線分と線分を共有するような線分を追加してはいけない。
  1. X座標またはY座標が等しい2つの点を結ぶ線分
  2. ある点と、既に平面に追加されている線分について、点から線分に下ろした垂線上の線分(垂線を線分上に下ろせない場合は不可)
- 線分をこれ以上追加できなくなったら終了

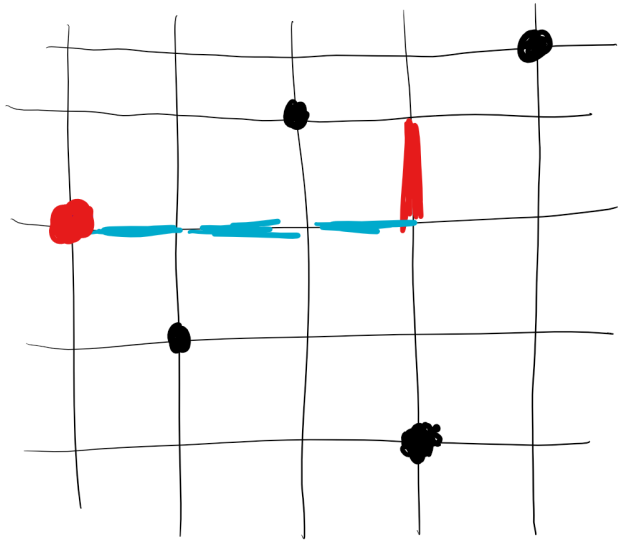
こうして追加した線分の長さの総和を求めよ

全探索

- とりあえず全探索をしてみよう
- すべての点、すべての追加された線分を毎回調べ、追加できる場合は追加する。
- 追加できなくなったら終了(ベルマンフォード法に似ている)
- 点の数はO(N)、線分の数は端点の通り数よりO(N^3)だが、座標圧縮をして線分を可能な限り小さく持つようにするとO(N^2)になる
- 更新回数は線分の数より、O(N^2)
- よって最悪計算量はO(N^4)
- 部分点はN≦100のため、適切な枝刈りで通るかもしれないし、通らないかもしれない
- （（そもそも更新回数O(N^2)というのはかなり詰めが甘い？？））

グラフ化

- 二次元平面状に出現する点のX座標、Y座標を座標圧縮し、縦横の長さO(N)の格子点として見る
- 点どうしを結ぶ操作は後から追加された線分に関係がないため最初からすべての点対どうしについて行ってしまう。(重要な考察！)
- 点と線分の垂線はどうか？
- 右図で、赤い線分が追加された場合、赤い点から下ろした垂線上の青い線分も追加される
- 各線分を頂点とした有向グラフのDFS(BFS)問題に帰着できる！

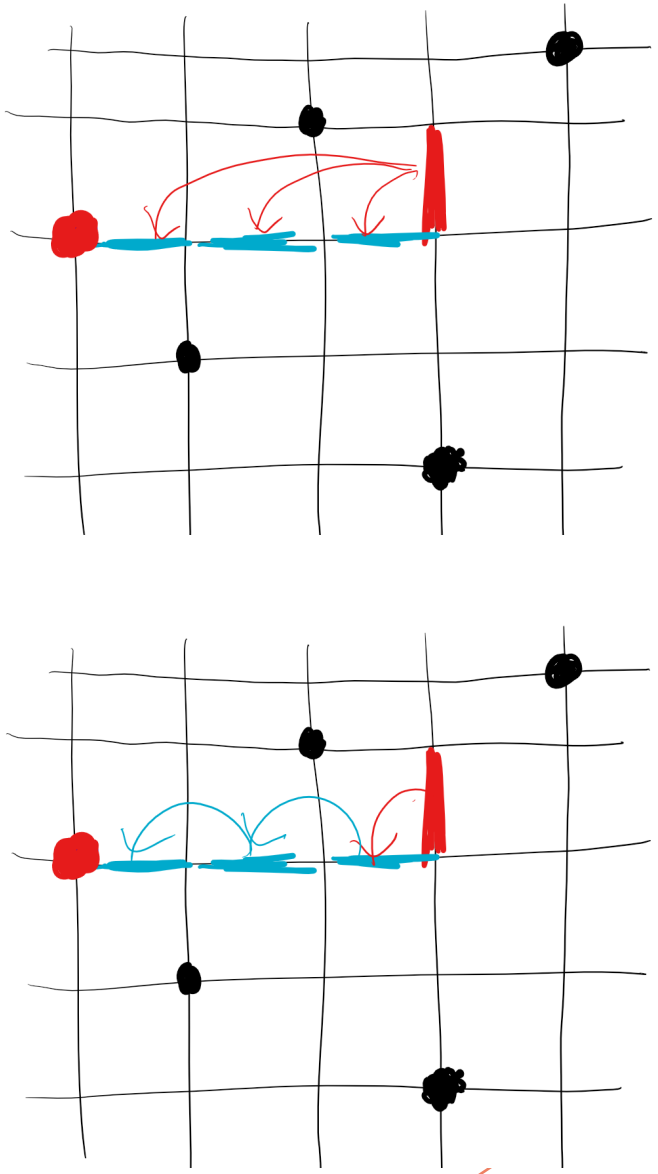


計算量

- 線分の個数はO(N^2)
- 線分どうしの関係を示す辺の個数は、線分はそれぞれO(N)個の線分と関係があるため、全体でO(N^3)
- 辺の数がO(N^3)のグラフをDFS(BFS)する時間計算量はO(N^3)
- N≦100には間に合うが、N≦1000は怪しい

辺のトリミング

- 右上の図の通り、各線分からO(N)の辺を伸ばす必要は実はない。
- 右下の図のように、赤い線分からは最寄りの青い線分へ辺を伸ばし、あとは青い線分同士を有向辺でつなげばよい。(辺の方向に注意)
- このように、グラフの辺を集約することによって計算量を落とすテクニックは覚えておくと便利。辺の伸ばし方に規則性がある場合、無駄な辺の張り方をしていないか注意するべきである。



計算量

- 赤い線分から青い線分への辺の数はO(N^2)に減った
- 青い線分同士の辺の数もO(N^2)
- 全体で、頂点数辺数ともにO(N^2)のグラフのDFS(BFS)問題に帰着できた。

- 計算量はO(N^2)のため、N≦1000で間に合う

終わり

- N≦100000はさすがに無理か…

終わり！？

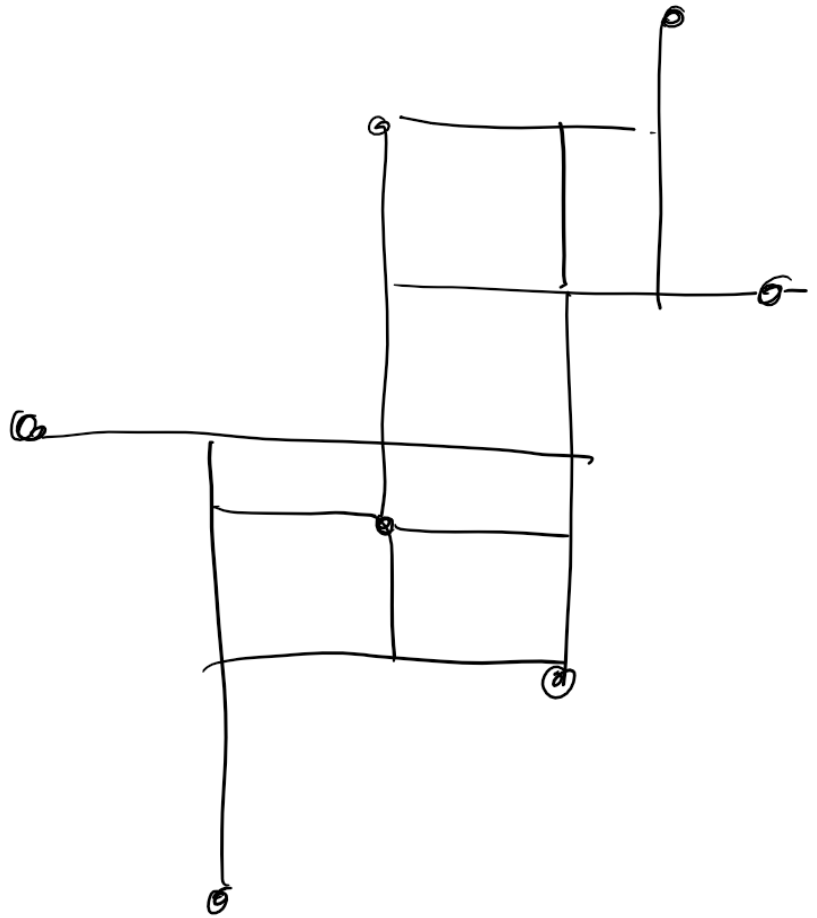
- N≦100000はさすがに無理か…

O(N log N)解
ありますよ？

！？

- 一度グリッドに分けるところから離れてみよう
- 改めて、線分でつながれた点の集合を見てみよう

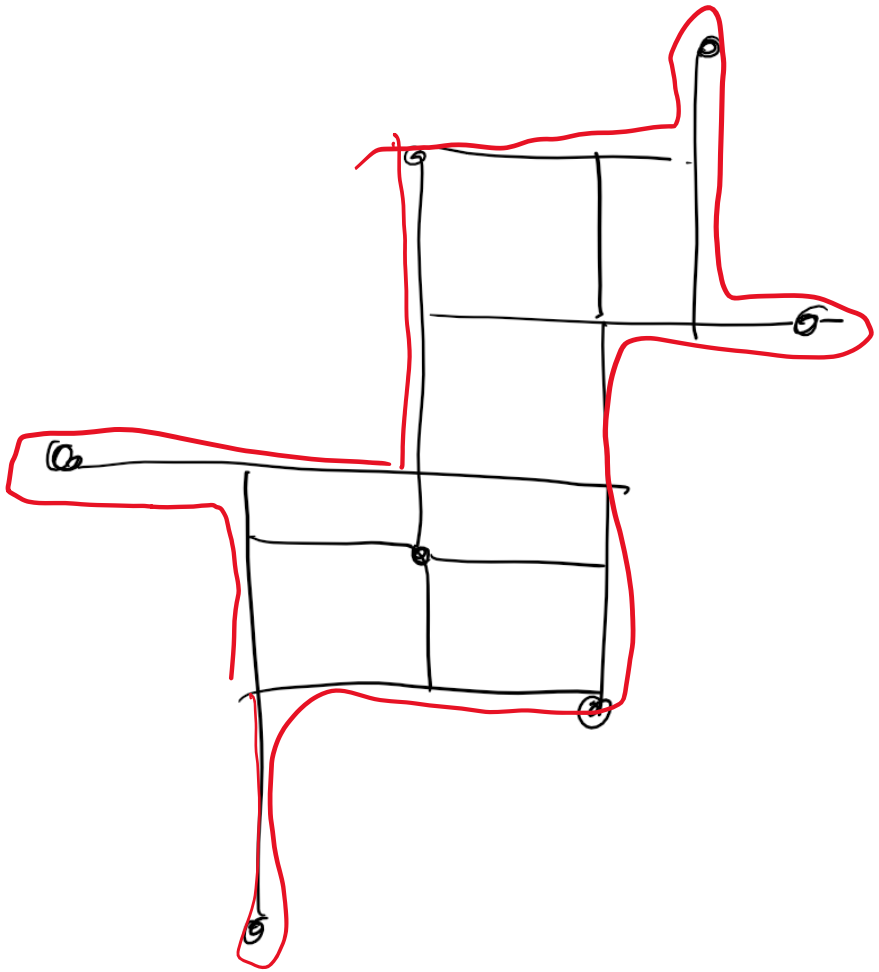
- 外側がいわゆる凸包のようになっている



！？

- 一度グリッドに分けるところから離れてみよう
- 改めて、線分でつながれた点の集合を見てみよう

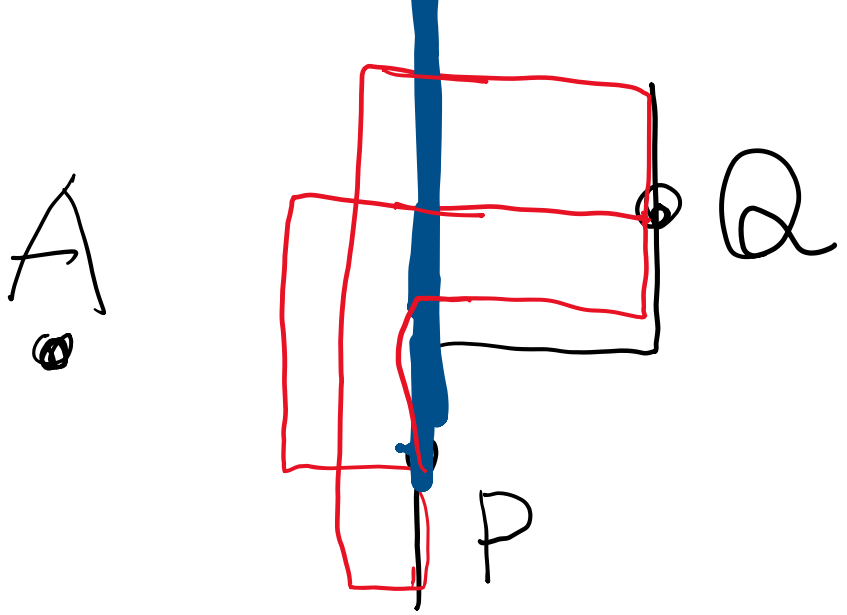
- 外側がいわゆる凸包のようになっている
- この外の赤い輪郭を把握できれば、各頂点から赤い線までめいっぱい伸ばした長さを計算することができ、答えを算出できるのでは？



連結成分の決定

- まずは、すべての頂点が、最終的に張られた線分をたどって行き来できると仮定する(**線分によって頂点が連結になっていると仮定する**)
- この時、ある点P(x,y)から右向き(x軸正方向)に伸びる線分は、
  x座標が
  min(y'≦yを満たすすべての点(x',y')の中で最も大きいx',
  y'≧yを満たすすべての点(x',y')の中で最も大きいx')
  まで伸びる！
- これは同様にして左向き、上向き、下向きについても計算できる。
- 重複する線分に注意すれば、これらの値の合計が、連結成分の頂点内で張られる線分の長さの総和になる。

証明



- では、なぜ、ある点(x,y)から
min(y’≦yを満たすすべての点(x’,y’)の中で最も大きいx’,
y’≧yを満たすすべての点(x’,y’)の中で最も大きいx’)
まで線分が伸びるといえるのか？
- 点(x,y)をAと置く。仮に、y’≦yを満たすすべての点(x’,y’)の中で最も大きいx’を持つ点をP、 y’≧yを満たすすべての点(x’,y’)の中で最も大きいx’を持つ点をQと置く。Pのx座標≦Qのx座標とする。(対称性が保証される)
- 仮定より、PとQは連結である。右上図より、PとQを繋ぐパス(赤い線)は、必ず「半直線x=Pのx座標,y≧Pのy座標(青い線)」を通過する。
- Pから半直線を通過する線分へ線分を下ろすことは可能である。この線分に向けてAから線分を下ろすことにより、Aから半直線まで線分を引くことができた。
- 同様の議論により、Aから半直線を越えて線分を引くことはできないことがわかる。

ここまでの計算量

- 点をn個含む連結成分について、
- 一つの点から四方に伸びていく線分の長さを知るには、点(x,y)について
  min(y’≦yを満たすすべての点(x’,y’)の中で最も大きいx’,
  y’≧yを満たすすべての点(x’,y’)の中で最も大きいx’)
  を計算する必要がある。
- これは、あらかじめx座標(or y座標)で点群をソートした後、y座標(or x座標)をセグメント木などのデータ構造に入れ、区間最小/最大問題に帰結すれば各処理をO(log n)で済ますことができる。
- 点の個数はO(n)なので全体でO(n log n)
- よって全体の点N個についてもその和でO(N log N)で計算できる

連結成分の判定

- では、どの点がどの点と連結であるかを知るにはどうすればよいか？
- 連結性を管理するときに便利なUnion find treeを使う
- ある点の集合(x1,y1),(x2,y2),(x3,y3),…,(xM,yM)を考えるときに、実は点一つ一つの座標を持っておく必要はない
- すべての点の中で最小のx(minx)、最大のx(maxx)、最小のy(miny)、最大のy(maxy)の4つの値さえ管理しておけば問題はない！
- なぜならば、その長方形内にどのように点が配置してあろうと、「minx≦x'≦maxxまたはminy≦y'≦maxyを満たす点(x',y')からは必ず線分を引くことができ、またそうでない点からは絶対に(それらの集合へ直接)線分を引くことができない」からである

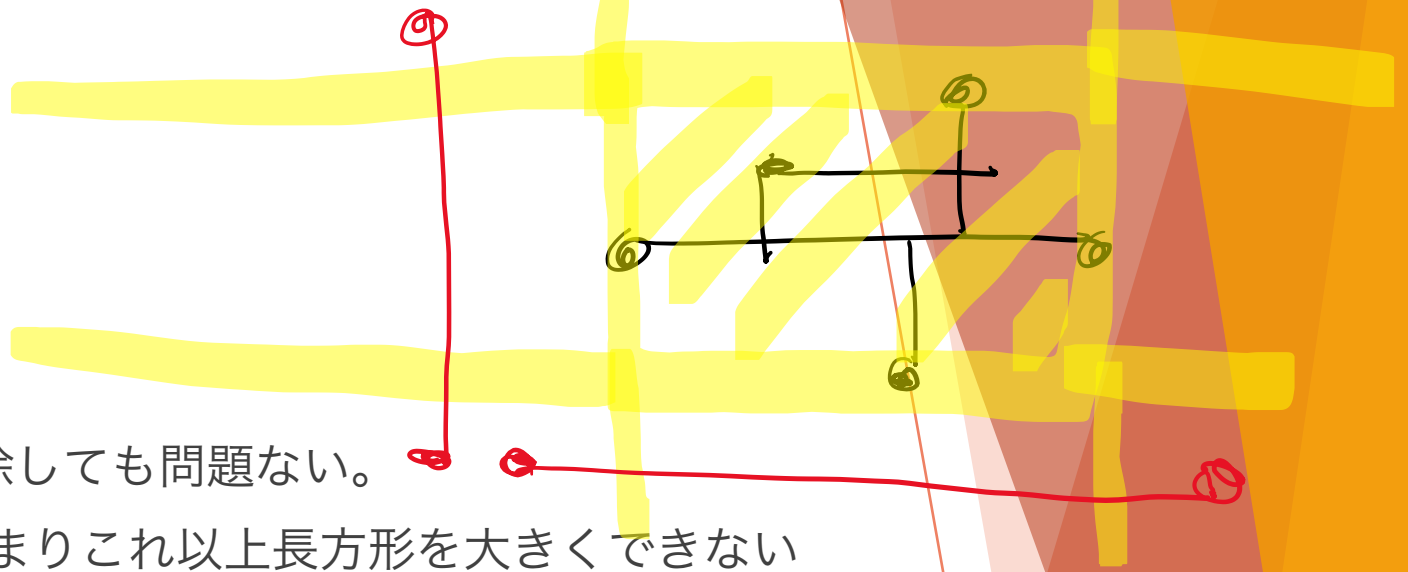
アルゴリズム

(*)長方形(p,q,r,s)とは、
p≦x≦q、r≦y≦sを満たす
点(x,y)の集合である

1. ある点(x,y)について、長方形(x,x,y,y)を与えておく
2. 長方形(minx,maxx,miny,maxy)を持っているとき、minx≦x'≦maxxまたは miny≦y'≦maxyを満たすすべての点(x',y')を連結判定とする。
3. 長方形を広げる。ステップ2.の条件を満たしたすべての点(x',y')について、以下のように代入更新を行う
   `minx=min(minx,x');maxx=max(maxx,x');miny=min(miny,y');maxy=max(maxy,y');`
4. 長方形をこれ以上広げられなくなるまでステップ2. 3.を続ける。

- 以上の処理をすべての点について行えば、連結判定を行うことができる
- が、これでは遅すぎる！(最悪計算量がO(N^2 * α(N))になる)

点を間引く



- 連結判定を行った後、いくつかの点は削除しても問題ない。
- 実は、連結判定をめいっぱい行った後(つまりこれ以上長方形を大きくできないところまで大きくした後)は、**最初の点を除きすべての点を削除してしまって構わない！**
- 理由は、これ以上長方形を大きくできないところまで大きくした後、さらにその外側に連結になる点が存在する
  ⇔外側の4隅のうち2隅を端点とする線分が伸びている
  ⇔一個だけ点を残しておけばそれらの点を確認するときに連結判定がされるからである。
- 以上の考察から、前述のアルゴリズムをこのように改良する

アルゴリズム

1. ある点(x,y)について、長方形(x,x,y,y)を与えておく
2. 長方形(minx,maxx,miny,maxy)を持っているとき、minx≦x’≦maxxまたはminy≦y’≦maxyを満たすすべての点(x’,y’)を連結判定とする。
3. 長方形を広げる。ステップ2.の条件を満たしたすべての点(x’,y’)について、以下のように代入更新を行う

   ```
   minx=min(minx,x’);maxx=max(maxx,x’);miny=min(miny,y’);maxy=max(maxy,y’);
   ```
4. 長方形をこれ以上広げられなくなるまでステップ2. 3.を続ける。
5. 最後に、**ステップ2の条件を満たしたすべての点を削除してしまう**(自分自身のみ残しておく)
6. ステップ1に戻る。

- 上の改良によって、計算量はどうなるだろうか？

計算量

- std::setなどの平衡二分木を２個持っておくことにより、点をO(log N)でx座標、y座標で検索、削除できるようにしておく。
- この時、ステップ2で削除されない一つの点を除き、原則としてすべての点は一度見た後消されてしまう。よって、全体で削除するときの計算量はO(N log N)になる。
- また、削除されない一つの点の影響を考えても、全体でみれば計算量O(N)しか変わらないことがわかる
- よってこのアルゴリズムは全体で計算量O(N log N)である

まとめ

- この問題では、「点を連結成分ごとに分ける」「連結成分ごとに線分の長さを算出する」の2パートに分かれている。
- 両方計算量はO(N log N)なので、全体でも計算量O(N log N)
- N≦100000でもこの問題を解くことができた

おわりに

- この問題はもともとN≦3000程度で出す予定でしたが、testerの某さんがO(N log N)解を提示してくださったことにより、かねての予定よりはるかに難しい問題になってしまいました。
- 平衡二分木、Union find tree、セグ木など、競プロでよくお目にかかるデータ構造を一通り使っているため、データ構造の総合力が試される
- 二次元座標平面の性質から、コピペを多用するかバグりやすいO(1)for文を利用することになるので、丁寧なコーディング・デバッグをする必要がある