AoA Programming Assignment 3

Author: Gabriel Hofer

Course: CSC-372

Instructor: Dr. Rebenitsch

Due: November 5, 2020

Computer Science and Engineering
South Dakota School of Mines and Technology

Returns the area enclosed by the polygon formed by convex hull.

```
typedef pair<double,double> pt;

double getArea(const vector<pt>& points) {
  double area=0;
  for (int i=0; i<points.size(); i++) {
    pt a = i ? points[i - 1] : points.back();
    pt b = points[i];
    area+= (a.first-b.first) * (a.second+b.second);
  }
  return fabs(area) / 2;
}
```

Monotone Chain algorithm for finding convex hull. Function ”cp” is the cross product of two vectors.

```
typedef pair<double,double> pt;

int cp(const pt a, const pt b, const pt c){
  return (b.first - a.first) * (c.second - a.second) -
    (b.second - a.second) * (c.first - a.first);
}

vector<pt> monotone(vector<pt> &pts){
  if(pts.size()<=3) return pts;
  vector<pt> L, U;
  sort(pts.begin(),pts.end());
  for(int i=0;i<pts.size();i++){
    while(L.size()>=2 && cp(L[L.size()-2], L.back(), pts[i]) <= 0)
      L.pop_back();
    L.push_back(pts[i]);
  }
  for(int i=pts.size()-1;i>=0;i--){
    while(U.size()>=2 && cp(U[U.size()-2], U.back(), pts[i]) <= 0)
      U.pop_back();
    U.push_back(pts[i]);
  }
  L.pop_back(); U.pop_back();
  L.insert(L.end(),U.begin(),U.end());
  return L;
}
```

Function ”despeckle” removes rogue pixels or ”noise”.

```
typedef pair<double,double> pt;

void despeckle(vector<pt> & pts, const double thresh){
  vector<pt> tmp, hull;
  set<pair<double,int>> mayRemove;
  double a1, a2;
  pair<bool,bool> ok={true,false};

  hull=monotone(pts);
  a1=getArea(hull);

  while(pts.size()>3 && ok.first){
    ok.first=false;
    for(int i=0;i<pts.size();i++){
      tmp=pts;
      tmp.erase(tmp.begin()+i);

      hull=monotone(tmp);
      a2=getArea(hull);

      if(1.0-(a2/a1)>=thresh){
        mayRemove.insert({a2,i});
        pts=tmp;
        a1=a2;
      }
    }
    if(mayRemove.size()){
      cout<<"Delete pixel at ("<<pts[mayRemove.begin()->second].first
        <<", "<<pts[mayRemove.begin()->second].second<<")"<<endl;
      pts.erase(pts.begin()+mayRemove.begin()->second);
      ok={true,true};
      mayRemove.clear();
    }
  }
  if(!ok.second) cout<<"No pixels deleted."<<endl;
}
```

Question 1

The recursive calls of the closest points function form a binary tree because the set of points is divided in half and two recursive calls are made: one for the right half and one for the left. After those calls have returned, we know the closest pair of points in each half of the set.

So, in order to make a dendogram, we simply need to show the structure of the recursive calls by creating a single node in each recursive function call.

Then we add edges connecting each parent call to each of their two children recursive calls. Also, note that the base case probably connects two actual points to each other, although this is slightly implementation dependent.

```
CLOSEST-PAIR-OF-POINTS:
1. Sort the points from smallest to largest x coordinate (X)
2. Sort the points from smallest to largest y coordinate (Y)
3. Evenly divide the set of points into "left" and "right" sets (both X and Y!)
4. Repeat division recursively until there are only 2 points
5. To merge, the closest pair of points is either the closest pair
   returned from both sides or is across the split boundary as follows:
  a. find all the Y points within d of the split line where d is
     the distance between the closest points from both halves.
  b. check the nearby points that could be closer (there are only 7 max)
  c. update the minimum pair
  // Dendogram Part
6. Create a Node
  a. Add an edge between this Node and the node returned from
     the left-half recursive call.
  b. Add an edge between this Node and the node returned from
     the right-half recursive call.
```

Question 2

a. [5 points] What is the optimal substructure (you must name the specific item being removed)?

The subproblems in this problem occur when we add some subset of rods in the total set of rods that we have available $SomeRods \subset AllRods$ to the beam. Also, in our algorithm, we always add the smallest rod to the beam. Thus, *SomeRods* will contain the first $SomeRods.size$ smallest rods from the set of all rods *AllRods*.

b. [6 points] What is the greedy property?

Greedy Choice Property: A global optimum can be attained by selecting a local optimum. For this problem, every time we add a rod to the beam we maintain that the current solution is optimal for the subset of rods that have already been added to the beam AND this locally optimal solution will be used to build more globally optimal solutions.

c. [9 points] Prove correctness of your algorithm. A proof of contradiction is acceptable.

```
MAKE-BEAM:
  layer:=1
  while remainingRods.size() > 0  do
    beam[ layer ].add(min(remainingRods))   // add shortest rod to beam
    remainingRods.erase(min(remainingRods)) // remove rod from list
    rodCnt[ layer ] += 1                    // count rods in this layer
    if remainingRods.size() <= rodCnt[ layer ] then
      beam[ layer ].add(remainingRods)      // add all rods to the beam
      rodCnt[ layer ] += remainingRods.size()
      remainingRods.clear()                 // remove all rods
      break while loop
    if rodCnt[ layer ] > rodCnt[ layer - 1 ]
      inc layer                             // goto next layer
```

While the length of the current layer is less than or equal to the length of the previous layer, add the shortest rod that hasn't been welded to a beam to the current layer. Every time we add a rod to the beam we also have to check whether there are not enough rods to make another layer after the current layer is completed. In order words, if $remainingRods.size() \leq rodCnt[layer]$ (i.e. if the number of rods that haven't been welded to the beam is less than or equal to the number of rods in the current layer), then we simply add all of the remaining rods to the current layer. We do this because otherwise, we would violate the criterion that each layer must have more layers than the previous layer. This will happen every time the number of rods in the set is not a triangle number.

As a reminder there are two loop invariants:

1. The total summed length of rods in the i^{th} layer must be less than $(i+1)^{th}$layer.
2. The total number of rodsin the i^{th} layer is less than the $(i+1)^{th}$ layer.

Initialization: at initialization, there are zero rods welded to the beam, **Maintenance: Termination:**

There are two cases, one for each criterion. First, assume that the number of rods in layer i is not greater than the sum of rods in layer i-1. This is impossible because

Question 4

Step	Add/Remove	Segments	Lines Compared	Intersection?
1	Add A	A	NA	N
2	Add B	B, A	A&B	N
3	Add C	B, A, C	A&B, A&C, B&C	N
4	Add D	B, D, A, C	A&B, A&C, A&D, B&C, B&D, C&D	N
5	Remove A	B, D, C	B&C, B&D, C&D	N
6	Remove B	D, C	C&D	N
7	Add E	D, C, E	C&D, C&E, D&E	N
8	Remove C	D, E	D&E	N
9	Remove D	E	NA	N
10	Add G	E, G	E&G	N
11	Add F	E, G, F	E&F, E&G, F&G	N
12	Add H	E, G, F, H	E&G, E&F, E&H, G&F, G&H, F&H	Y

Question 3