CEOI'2012 Day1, Task: circuit

The model solution of the task circuit is an $\mathbf{O}(n \cdot \log n)$ running time algorithm, where n is the number of nodes of the circuit.
The circuit is a closed polygon given by the sequence of the nodes P, and the location of each node is given by x- and y-coordinates. The segments of the polygon are $\overrightarrow{P[i], P[i+1]}$, $i < n$ and $\overrightarrow{P[n], P[1]}$. The task asks to compute all nodes of the polygon that are visible from the origin.
The basic operation used by the algorithm is $Turn(a, b, c)$. $Turn$ returns 0 if the points $P[a], P[b]$ and $P[c]$ are collinear, otherwise it returns 1 if line segment $\overrightarrow{P[b], P[c]}$ turns left at point $P[b]$, and returns -1 if turns right. Let us denote the origin by 0 and use $Turn0(a, b)$ for $Turn(0, a, b)$.
Let $first$ is the node for which any other node u $Turn0(first, u) > 0$ or $Turn0(first, u) = 0$ and $P[first].x < P[u].x$. Similarly, let $last$ is the node for which any other node u $Turn0(last, u) < 0$ or $Turn0(last, u) = 0$ and $P[last].x < P[u].x$.
It is clear that only the lower part of the polygon which comes from the node $first$ to the node $last$ is relevant

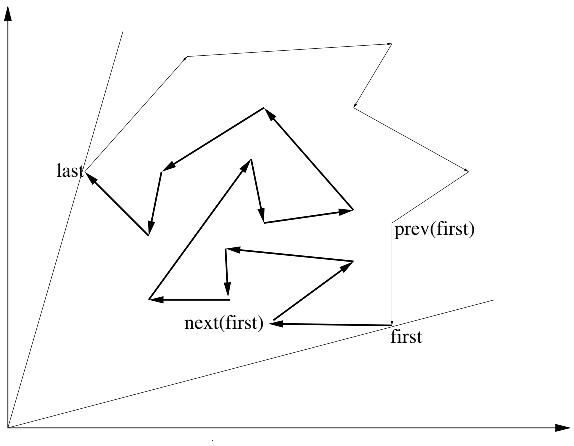

Figure 1:

because the nodes on the upper part are not visible. In the rest of the discussion we consider only the nodes of the lower part. The numbering of the nodes from $first$ to $last$ is either increasing or decreasing, we use the function $next(i)$ to refer to the node following the node i. Similarly $prev(i)$ refers to the previous node.
It is easy to prove that if for any none a $Turn0(a, next(a)) \geq 0$ then node a can not be visible. See figure 2. Moreover if node a is visible then $Turn0(Prev(a), a) > 0$.
Therefore we can consider only those segments for which $Turn0(a, next(a)) > 0$.

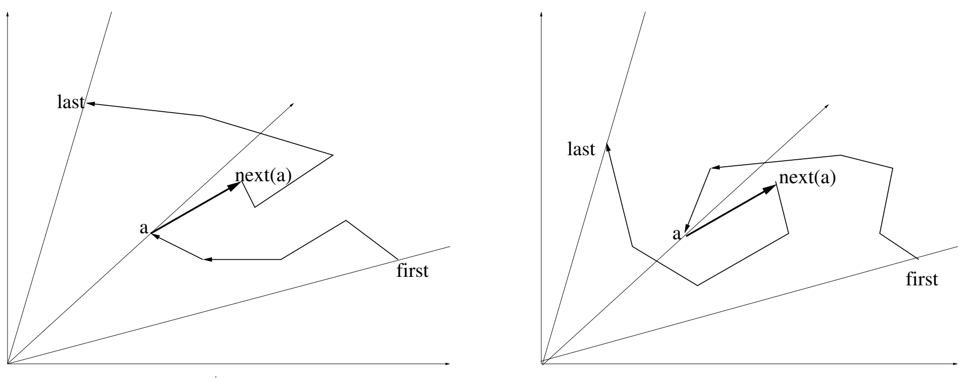

Figure 2:

For each node consider the set $Q(i)$:

$$Q(i) = \{j : Turn0(i, j) \leq 0 \text{ and } Turn0(i, next(j)) \geq 0\}$$

We define the relation "below" on the set $Q(i)$ as follows:

$$u \text{ below } v \Leftrightarrow \text{Turn0}(u,v) \geq 0 \text{ and Turn(u,next(u),v) } \leq 0 \text{ or } \text{Turn0}(u,v) < 0 \text{ and Turn(v,next(v),u) } > 0$$

In other words u below v if and only if the intersection of the line $l(0,i)$ and segment $\overrightarrow{u, next(u)}$ is closer to the origin then the intersection of the line $l(0,i)$ and segment $\overrightarrow{v, next(v)}$.
One can see that the relation "below" is a linear ordering relation on the set $Q(i)$. Using this property of the

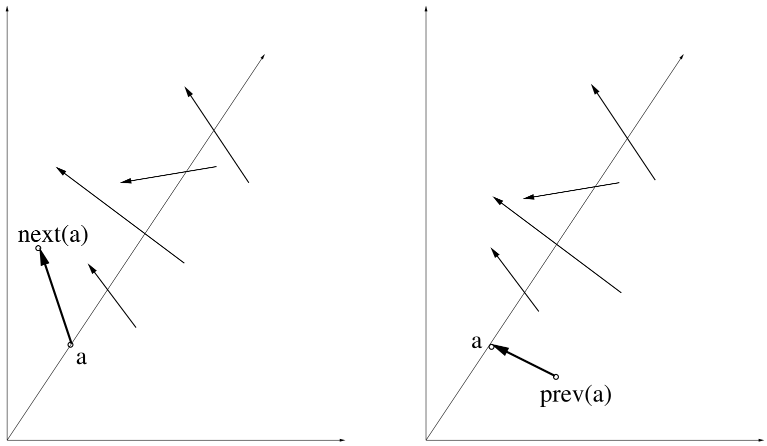

Figure 3:

relation we can state or main proposition. (Remember that $i \in Q(i)$.)

Proposition

Assume that either $Turn0(a, next(a)) > 0$ or $Turn0(prev(a), a) > 0$ holds for node a. Then node a is visible form the origin if and only if a is the least element of $Q(a)$ according to the relation "below".
The proof is straightforward.
The algorithm constructs the set $Q(a)$ for all nodes a for which $Turn0(a, next(a)) > 0$ holds and observe whether a is the least element in $Q(a)$. The sets $Q(a)$ will be constructed incrementally in the following way. Sort the nodes (that are between $first$ and $last$) according to polar angle. Let a be the next node that follows node $a0$ according to the polar angle ordering. Then $Q(a)$ is obtained from $Q(a0)$ as follows.
First, if prev$(a) \in Q(a0)$ then delete prev(a) from $Q(a0)$.
Prior to deletion test if $prev(a)$ is the first element in $Q(a0)$ and if so then add a to the solution set.

$$Q(a) = Q(a0) - \{\text{prev}(a)\}$$

Then if $\text{Turn0}(a, \text{next}(a)) > 0$ add a to $Q(a0)$

$$Q(a) = Q(a0) \cup \{a\}$$

If a is the first element in $Q(a)$ after the insertion then add a to the solution set.
The required operations can be implemented efficiently by binary heap that supports deletion. Implementation of deletion is obtained by using a map to the heap, if HT is the array of the heap we use the array Map such that $HT[Map[i]] = i$ and $Map[HT[j]] = j$. By this implementation the running time of addition and deletion is $O(\log k)$ in worst case if the set contains k elements. Observing the least element in $Q(a)$ is constant time operation. Using quicksort for the polar angle sorting we obtain an $\mathbf{O}(n \cdot \log n)$ running time algorithm.

Implementation

```
#include <stdlib.h>
#include <stdio.h>
#define  maxN 200001
using namespace std;
typedef struct Point{
    long x,y;
    bool act;
}Point;

typedef struct MPQ{
    int e_num;      //number of elements in the priority queue
    int HT[maxN];   //the binary heap tree
    int Map[maxN];  //the map into the heap: HT[Map[i]]==i, Map[HT[j]]==j
}MPQ;

MPQ Q;
Point P[maxN]; //the input polygon
int Ps[maxN];  //sorted points
bool Sol[maxN];//solution nodes
int n,d;
//priority queue operations:
void Push(int id);
int Top();
void Delete(int id);
void pushdown(int k);
void lift(int k);

int next(int i){
    i+=d;
    if (i>n) return 1;
    if (i==0) return n;
    return i;
}
int prev(int i){
    i+=(-d);
    if (i>n) return 1;
    if (i==0) return n;
    return i;
}
int Turn(int P0,int P1,int P2){
    long long crossp=(long long)(P[P1].x-P[P0].x)*(P[P2].y-P[P0].y)
                    -(long long)(P[P2].x-P[P0].x)*(P[P1].y-P[P0].y);
    if (crossp<0)
        return -1;
    else if (crossp>0)
        return 1;
    else
        return 0;
}
int Turn0(int P1,int P2){
    long long crossp=(long long)(P[P1].x)*(P[P2].y)
                    -(long long)(P[P2].x)*(P[P1].y);
    if (crossp<0)
        return -1;
    else if (crossp>0)
        return 1;
    else
        return 0;
}
```

```
bool below(int i, int j){
//segment (i,next(i)) is below segment (j,next(j))
    if (Turn0(i,j)>=0)
        return Turn(i,next(i),j)<=0;
    else
        return Turn(j,next(j),i)>0;
}
//implementation of the priority queue operations by binary heap
//Q is gobal for the operations
void Push(int id) {
  Q.HT[++Q.e_num] = id;
  Q.Map[id] = Q.e_num;
  if (Q.e_num>1) lift(Q.e_num);
}
int Top() {
  if (Q.e_num == 0) return 0;
  return Q.HT[1];
}
void Delete(int id){
  int k = Q.Map[id];
   Q.HT[k]=Q.HT[Q.e_num];
   Q.e_num--;
    if (Q.e_num>1) pushdown(k);
}
void pushdown(int k){
  int parent = k;
  int sun;
  int id = Q.HT[k];
  while ((sun = parent << 1) <= Q.e_num ){
        if (sun < Q.e_num && below(Q.HT[sun + 1], Q.HT[sun]) )
               ++sun;
           if (below(Q.HT[sun], id)){
               Q.HT[parent] = Q.HT[sun];
               Q.Map[Q.HT[parent]] = parent;
               parent = sun;
           } else break;
     }
     Q.HT[parent] = id;
     Q.Map[id] = parent;
}
void lift(int k){
  int id = Q.HT[k];
  int sun = k;
  int parent;
  while (sun > 1) {
     parent = sun >> 1;
     if (below(id, Q.HT[parent]) ){
        Q.HT[sun] = Q.HT[parent];
        Q.Map[Q.HT[parent]] = sun;
        sun = parent;
     }
     else
        break;
  }
  Q.HT[sun] = id;
  Q.Map[id] = sun;
}
```

```
//order relation for sorting according to polar angle
int ord_rel(const void* a, const void* b) {
   int i =*(int*) a;  int j =*(int*) b;
   int tij=Turn0(i,j);
   if (tij>0 || tij==0 && P[i].x<P[j].x)
      return -1;
   else
      return 1;
}
```

```
int main(){
    long x,y;
    int first=1,last=1,aa,fab,m=1;
    scanf("%d",&n);
    scanf("%ld %ld",&x,&y);
    P[1].x=x; P[1].y=y;P[1].act=false;
    Sol[1]=false;
    for (int i=2;i<=n;i++) {
        scanf("%ld %ld",&x,&y);
        P[i].x=x; P[i].y=y; P[i].act=false;
        Sol[i]=false;
        fab=Turn0(first,i);
        if (fab<0 || fab==0 && P[i].x<P[first].x) first=i;
        fab=Turn0(last,i);
        if (fab>0 || fab==0 && P[i].x<P[last].x) last=i;
    }
    int a=first+1; if (a>n) a=1;
    int b=first-1; if (b==0) b=n;
    if (Turn(first,a,b)<0) d=1; else d=-1;
    if (next(first)==last){
        printf("2\n");
        if (a<first)
            printf("%d %d\n",a,first);
        else
            printf("%d %d\n",first, a);
        exit(0);
    }
    int nn=0; a=first;
    do{
        Ps[nn++]=a;a=next(a);
    }while(a!=next(last));
    qsort((char *)Ps, nn, sizeof(int),  ord_rel);
    Sol[first]=true;
    P[first].act=true;
    Q.e_num=0; Push(first);
    for (int i=1;i<nn;i++){
        a=Ps[i];
        aa=prev(a);
        if (P[aa].act){
            if (aa==Top() && Turn0(a,Ps[i-1])!=0){
                Sol[a]=true; m++;
            }
            Delete(aa);
            P[aa].act=false;
        }
        aa=next(a);
        if (Turn0(a,aa)>0){
            P[a].act=true;
            Push(a);
            if (Top()==a && Turn0(a,Ps[i-1])!=0 && !Sol[a])
                {Sol[a]=true; m++;}
        }
    }
    if (!Sol[last]){Sol[last]=true; m++;}
    printf("%d\n",m);
    for (int i=1;i<=n;i++) if (Sol[i]) printf("%d ",i);
    printf("\n");
    return 0;
}
```

We present here a liniar time solution without proof.

```
program circuit;

type int=longint;
const MAXN=1000000;
var
  x,y,z:array[0..MAXN] of int;
  n,d,w:int;

procedure inp;
var
  i:int;
begin
  x[0]:=0; y[0]:=0;
  readln(n);
  for i:=1 to n do readln(x[i],y[i]);
end{inp};

function Turn(a,b,c:int) : int;          //turn direction
var
  r:int64;
begin
  r:=(x[b]-x[a])*(y[c]-y[a])-(y[b]-y[a])*(x[c]-x[a]);
  if r>0 then Turn:=1 else if r=0 then Turn :=0 else Turn:=-1;
end{Turn};

function szm(a,b,e,f:int):boolean;      //segments intersect
begin
  if (Turn(a,b,e)<>Turn(a,b,f)) and (Turn(e,f,a)<>Turn(e,f,b)) then
    szm:=true else szm:=false;
end{szm};

procedure init;
var
  i,s,t,f:int;
begin
  t:=1;                                 //find minimum angle
  for i:=2 to n do begin
    f:=Turn(0,t,i);
    if (f=-1) or ((f=0) and (x[i]<x[t])) then t:=i;
  end;
  w:=1; z[w]:=t;                        //initialize stack

  s:=t-1; inc(t);                       //find direction
  if s=0 then s:=n;
  if t=n+1 then t:=1;
  d:=Turn(t,z[w],s);
end{init};
```

```
procedure proc;
var
  i,u,s,t:int;
  hid,h1,h2:int;
begin{proc}
  s:=z[1]; t:=z[1]+d; hid:=0;
  if t=0 then t:=n else if t=n+1 then t:=1;
  inc(w); z[w]:=t;
  for i:=2 to n-1 do begin
    u:=s; s:=t; t:=s+d;
    if t=0 then t:=n else if t=n+1 then t:=1;
    if hid=1 then begin              //s hidden behind a stalactite
      if Turn(0,z[w],t)<0 then begin
        hid:=0; dec(w);
        while (Turn(0,z[w],t)<=0) and (Turn(s,t,z[w])>0) do
          dec(w);
        if Turn(s,t,z[w])<0 then begin
          hid:=3; h1:=t; h2:=0;
        end else begin
          inc(w); z[w]:=t;
        end;
      end;
    end else if hid=0 then begin         //s not hidden  (here s=z[w])
      if Turn(0,z[w],t)>0 then begin
        if (Turn(0,u,s)<0) and (Turn(u,s,t)>0) then hid:=1
          else begin
            inc(w); z[w]:=t;
          end;
      end else begin
          if (Turn(0,u,s)>0) and (Turn(u,s,t)<0) then
            hid:=2
          else begin
            dec(w);
            while (Turn(0,z[w],t)<=0) and (Turn(s,t,z[w])>0) do
              dec(w);
            if Turn(s,t,z[w])<0 then begin
              hid:=3; h1:=t; h2:=0;
            end else begin
              inc(w); z[w]:=t;
            end;
          end;
      end;
    end else if hid=2 then begin         //s hidden in the background
      if Turn(0,z[w],t)>0 then begin
        hid:=0;
        inc(w); z[w]:=t;
      end;
    end else begin                        //s hidden behind a stalagmite
      if szm(z[w],h1,s,t) then h2:=1;
      if Turn(z[w],h1,t)<0 then h2:=0;
      if (h1=s) and (Turn(u,s,t)>0) then h2:=0;
      if (Turn(0,z[w],t)>0) and (h2=1) then begin
        hid:=0;
        inc(w); z[w]:=t;
      end;
    end;
  end;
end{proc};
```

```
procedure outp;
var
  i:int;
begin
  for i:=1 to n do x[i]:=0;
  for i:=1 to w do x[z[i]]:=1;
  writeln(w);
  for i:=1 to n do if x[i]=1 then write(i,' ');
  writeln;
end{outp};

begin{program}
  inp;
  init;
  proc;
  outp;
end.
```