POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Applicazioni degli Ordinamenti: l'Inviluppo Convesso

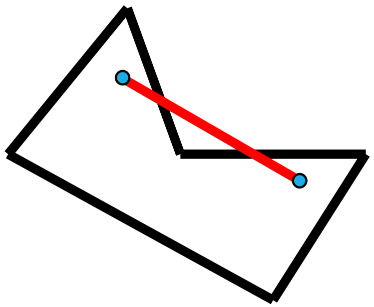
Geometria computazionale

Branca dell’Informatica che studia gli algoritmi atti a risolvere problemi geometrici:

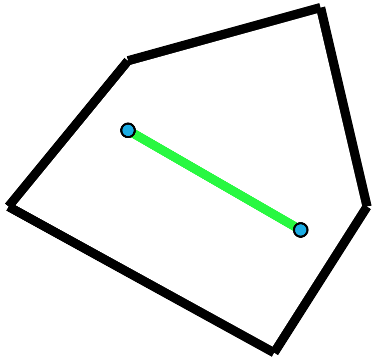
- intersezione di una coppia qualsiasi di segmenti
- inviluppo convesso
- ricerca della coppia di punti più vicini.

Poligono convesso

Un poligono si dice convesso se ogni segmento che congiunge due punti del poligono è interno al poligono stesso, incluso il bordo.



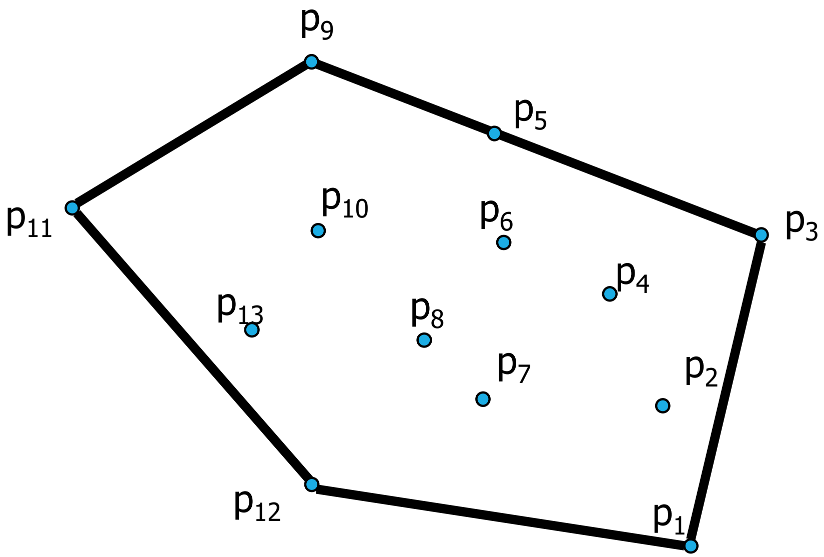
poligono concavo



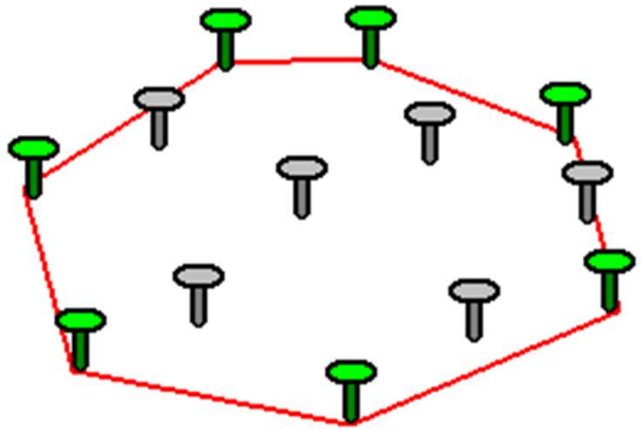
poligono convesso

Inviluppo (involucro) convesso – Convex hull

Dato un insieme di punti Q, l'inviluppo convesso è il poligono convesso P di area minima per cui ogni punto di Q è interno a P o al più sul perimetro di P:



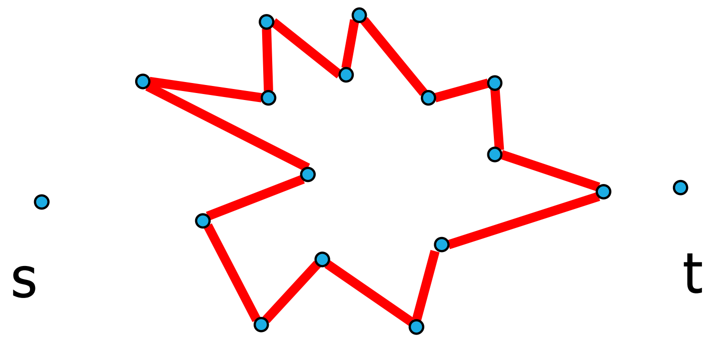
Soluzione «meccanica»: piantare chiodi perpendicolari al piano in ogni punto, racchiuderli con un elastico:



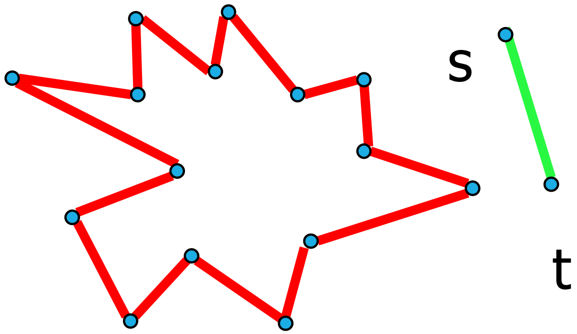
http://www.idlcoyote.com/math_tips/convexhull.html

Applicazione

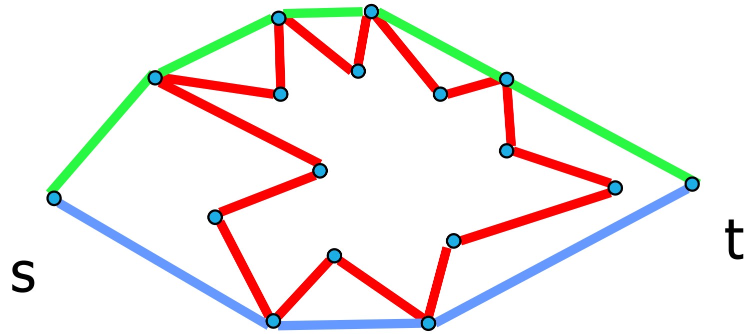
Pianificazione del moto di un robot con un cammino a lunghezza minima da un punto s a un punto t per evitare un ostacolo di forma poligonale:



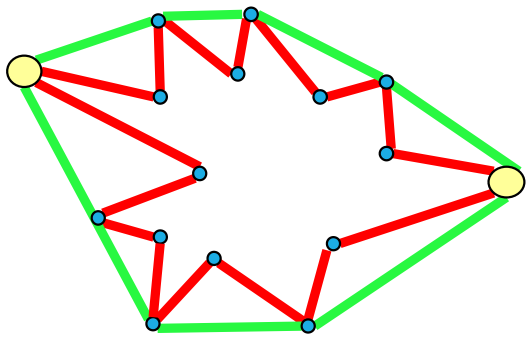
- o c’è un segmento s-t che evita l’ostacolo



- o è una delle 2 spezzate tra s e t lungo il contorno dell’inviluppo convesso



Dati N punti sul piano, trovare la coppia a distanza massima. Si dimostra che i 2 punti sono vertici dell’inviluppo convesso e che li si può trovare con complessità T(N) = O(N).



Soluzione brute-force

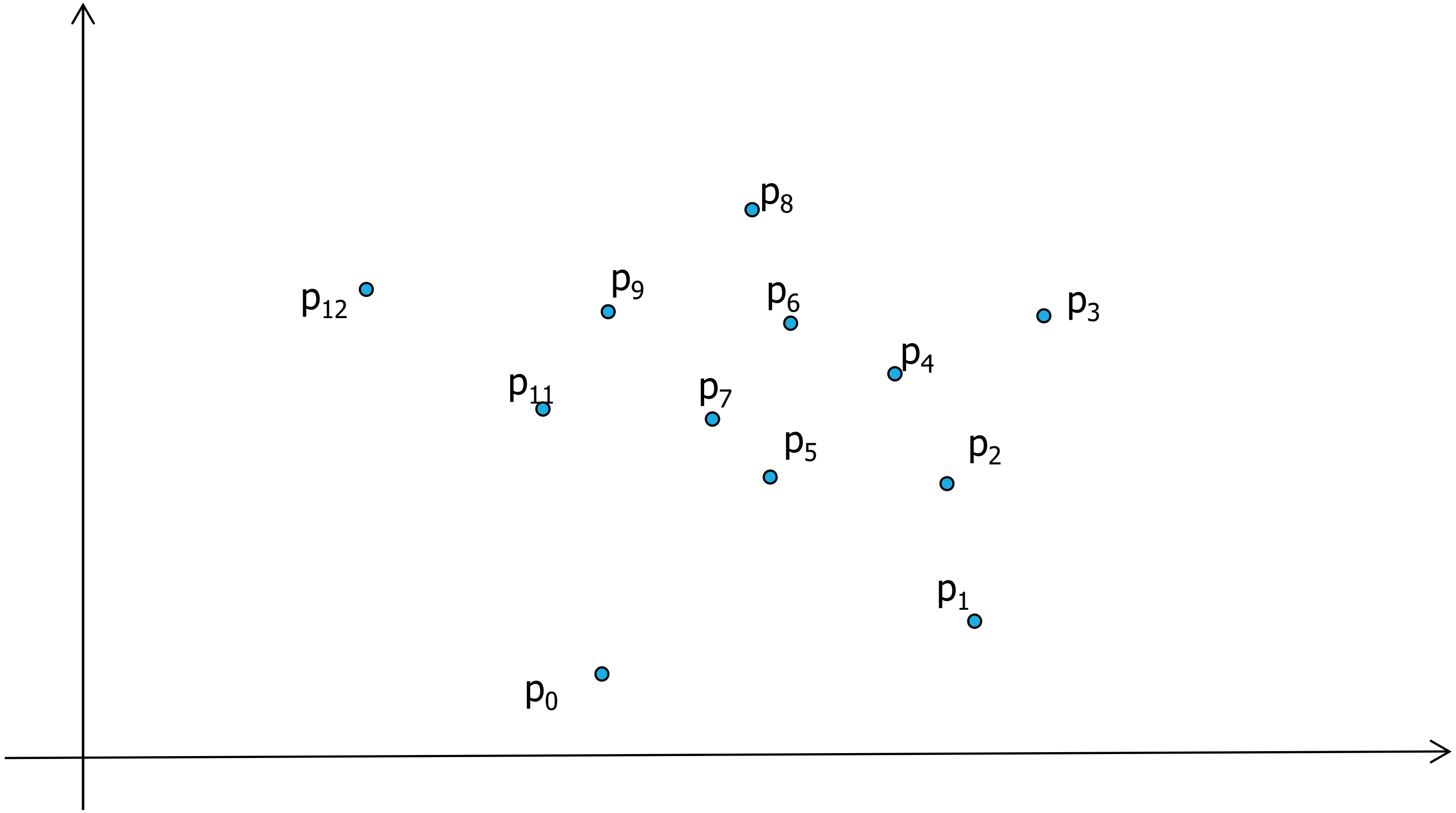
- dato l'insieme dei punti, costruire l'insieme dei suoi sottoinsiemi (insieme delle parti)
- per ciascun sottoinsieme verificare che si tratti di un poligono convesso e se sì calcolarne l'area
- tenere traccia dell'area minima
- complessità esponenziale.

Graham Scan (1972)

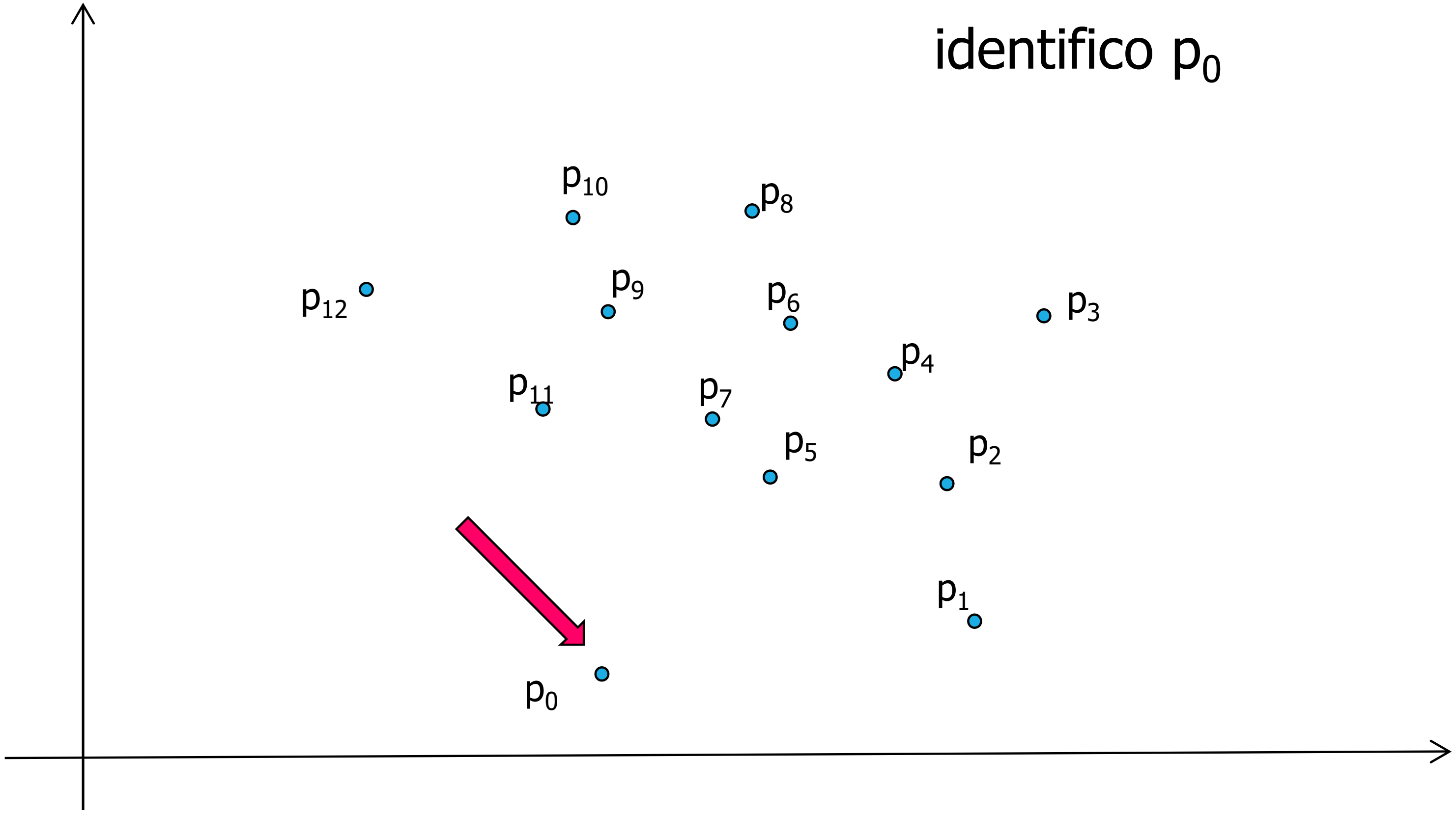
Dato un insieme di punti sul piano $p_0 \ldots p_N$

- identificare il punto a ordinata minima (il più a SX in caso di parità) e memorizzalo come p_0
- ordinare i punti rimanenti $p_1 \ldots p_N$ per angolo polare crescente rispetto a p_0. Per punti sulla stessa retta, tenere solo il più distante da p_0
- push sullo stack p_0, p_1, p_2
- per i rimanenti punti con i = 3 a N
  - fintanto che l'angolo formato dal punto sotto la cima dello stack, da quello in cima allo stack e da p_i non provoca una svolta antioraria (a sinistra), pop dallo stack
  - push sullo stack di p_i
- lo stack alla fine contiene i punti dell'inviluppo complesso in ordine antiorario

Esempio

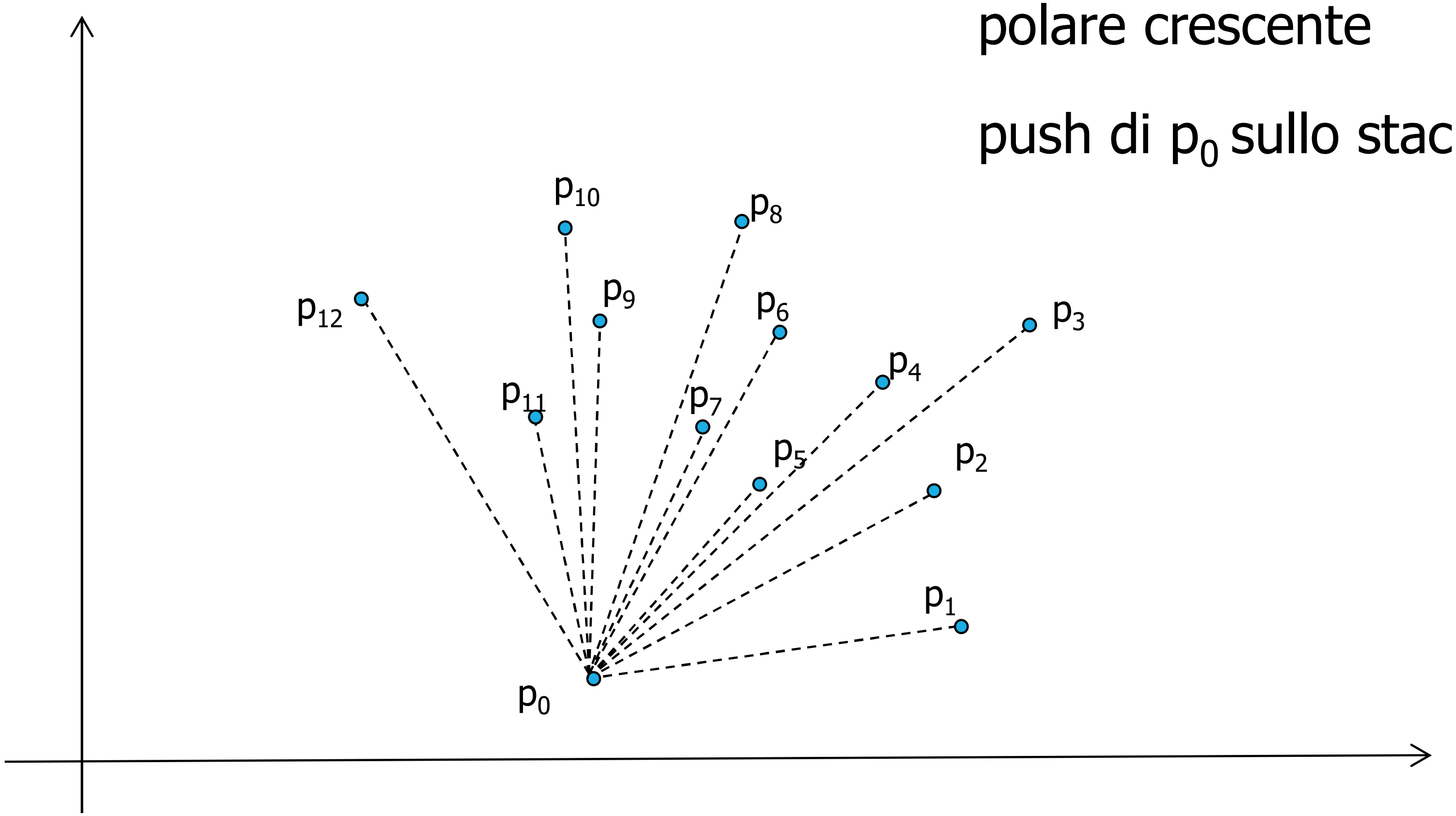


identifico p_0

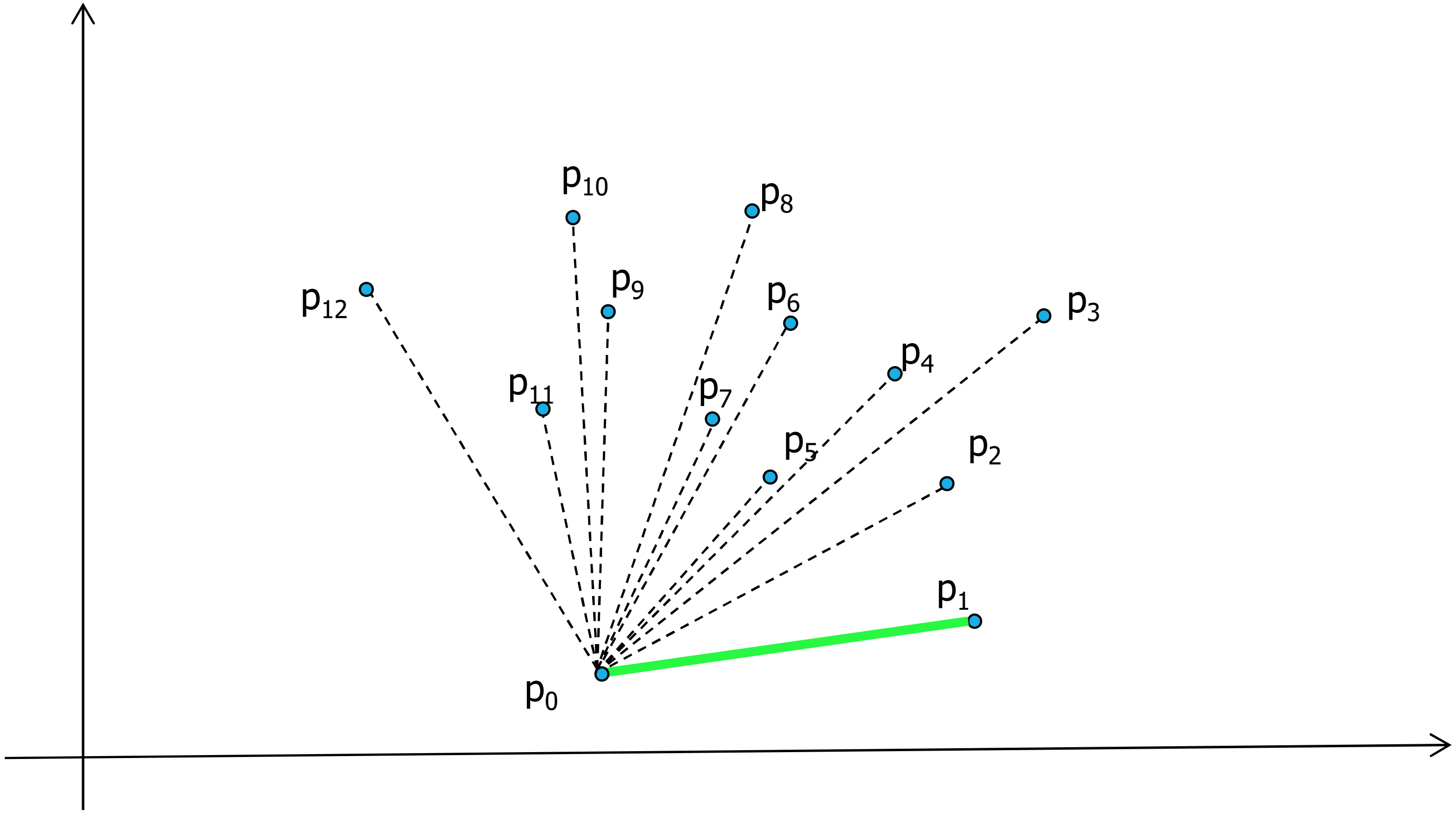


ordino i rimanenti punti per angolo polare crescente

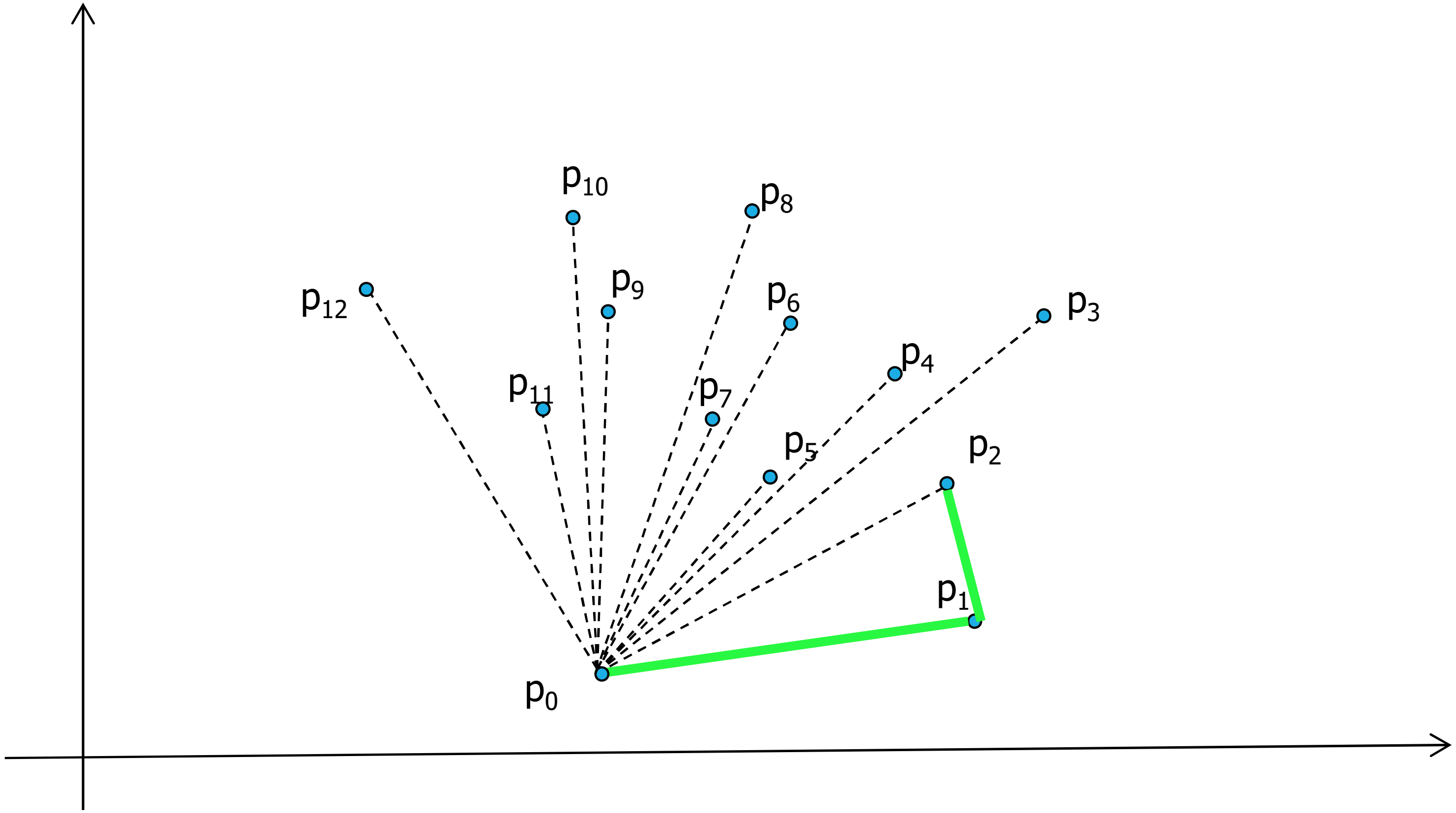
push di p_0 sullo stack



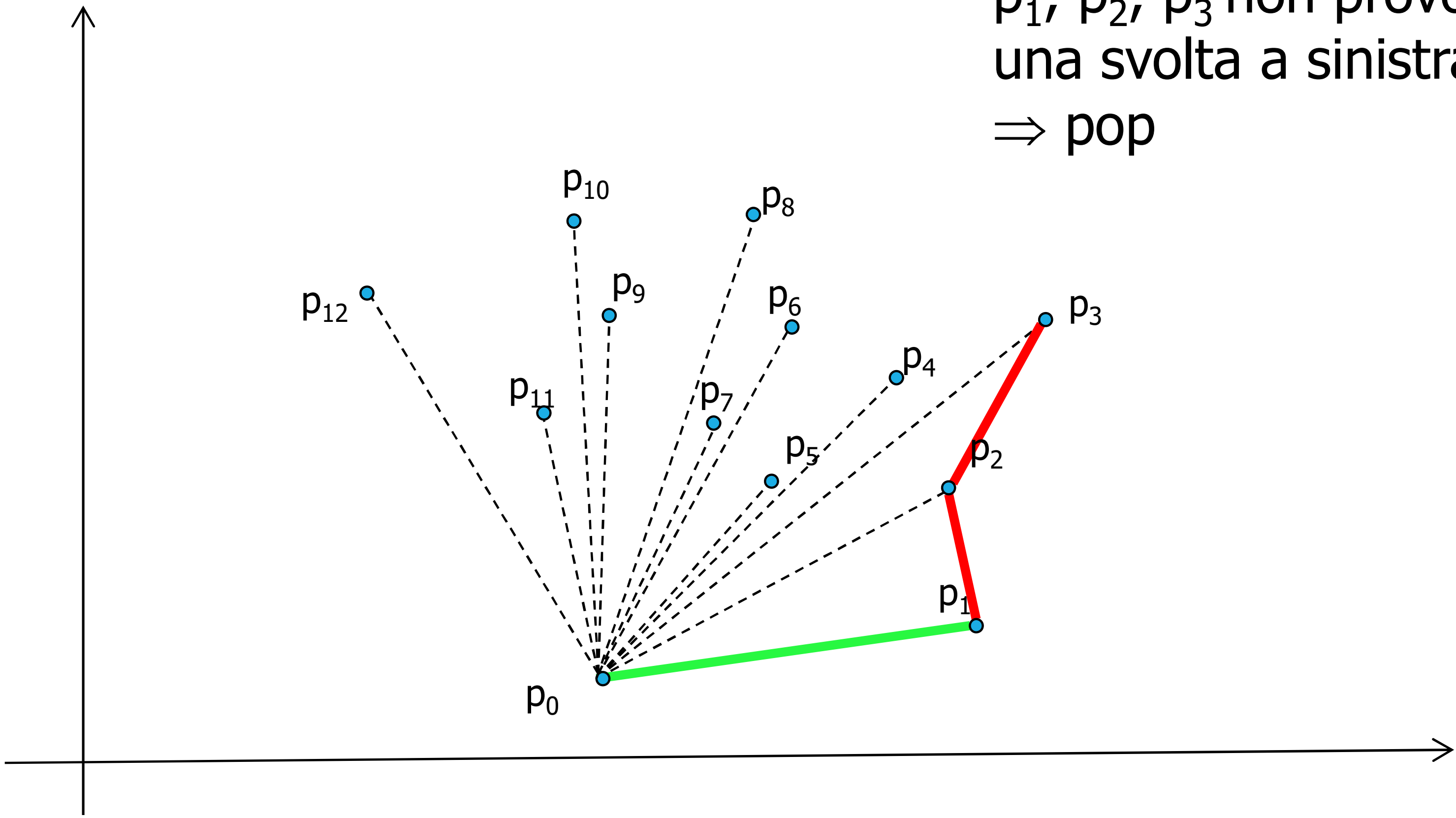
push di p_1 sullo stack

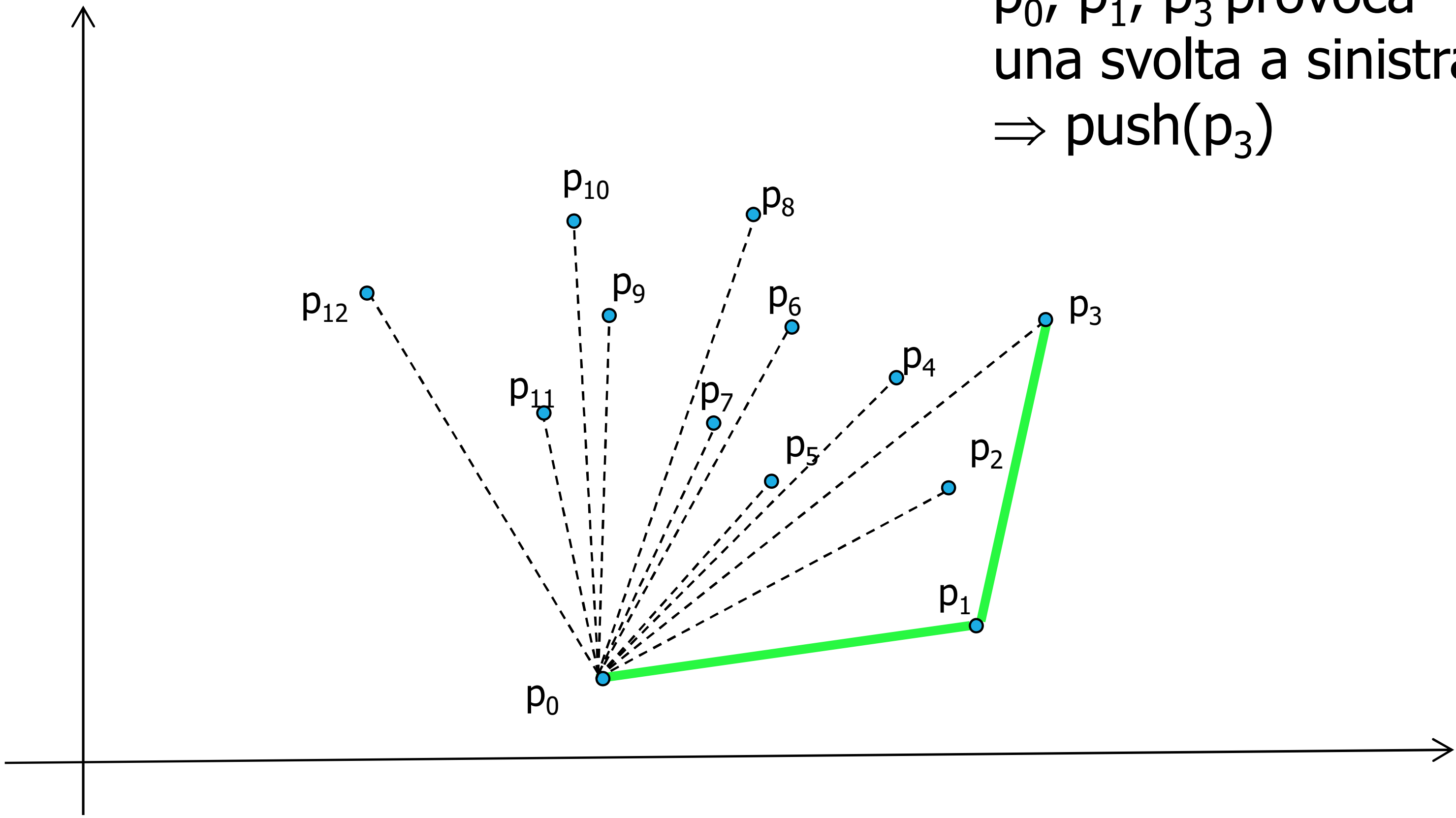


push di p_1 sullo stack



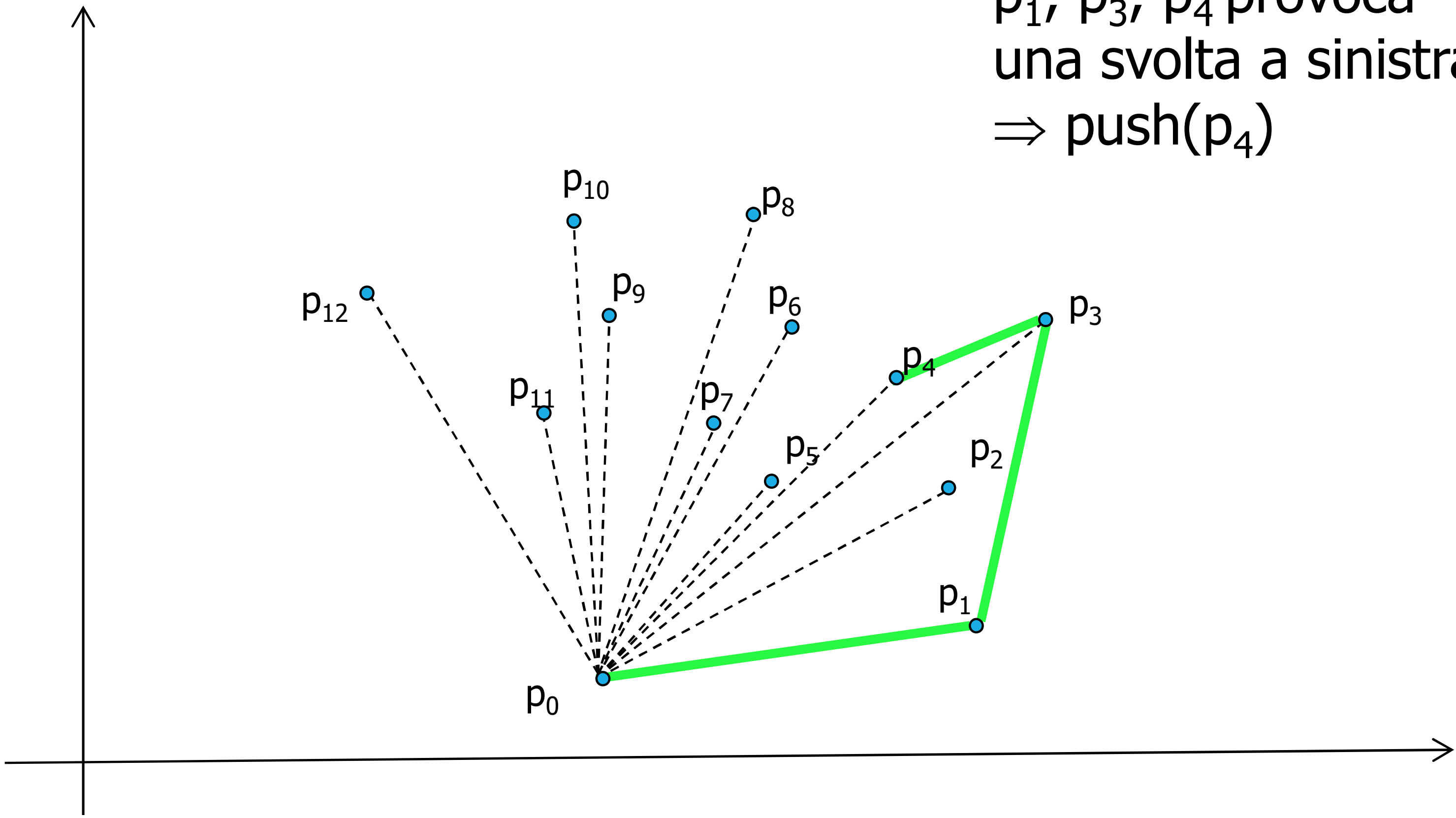
l'angolo formato da p_1, p_2, p_3 non provoca una svolta a sinistra $\Rightarrow$ pop



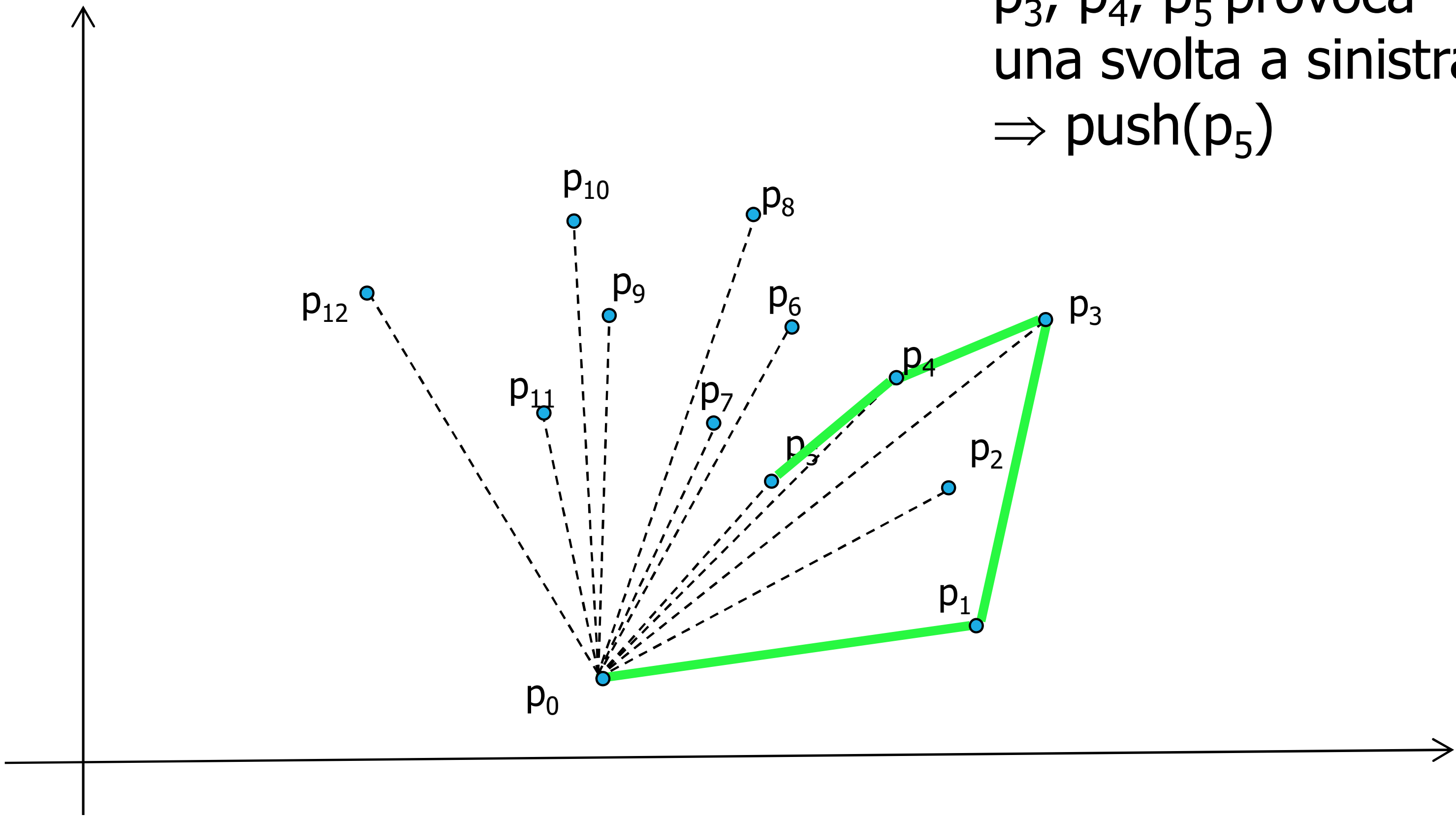


l'angolo formato da p_0, p_1, p_3 provoca una svolta a sinistra
$\Rightarrow$ push(p_3)

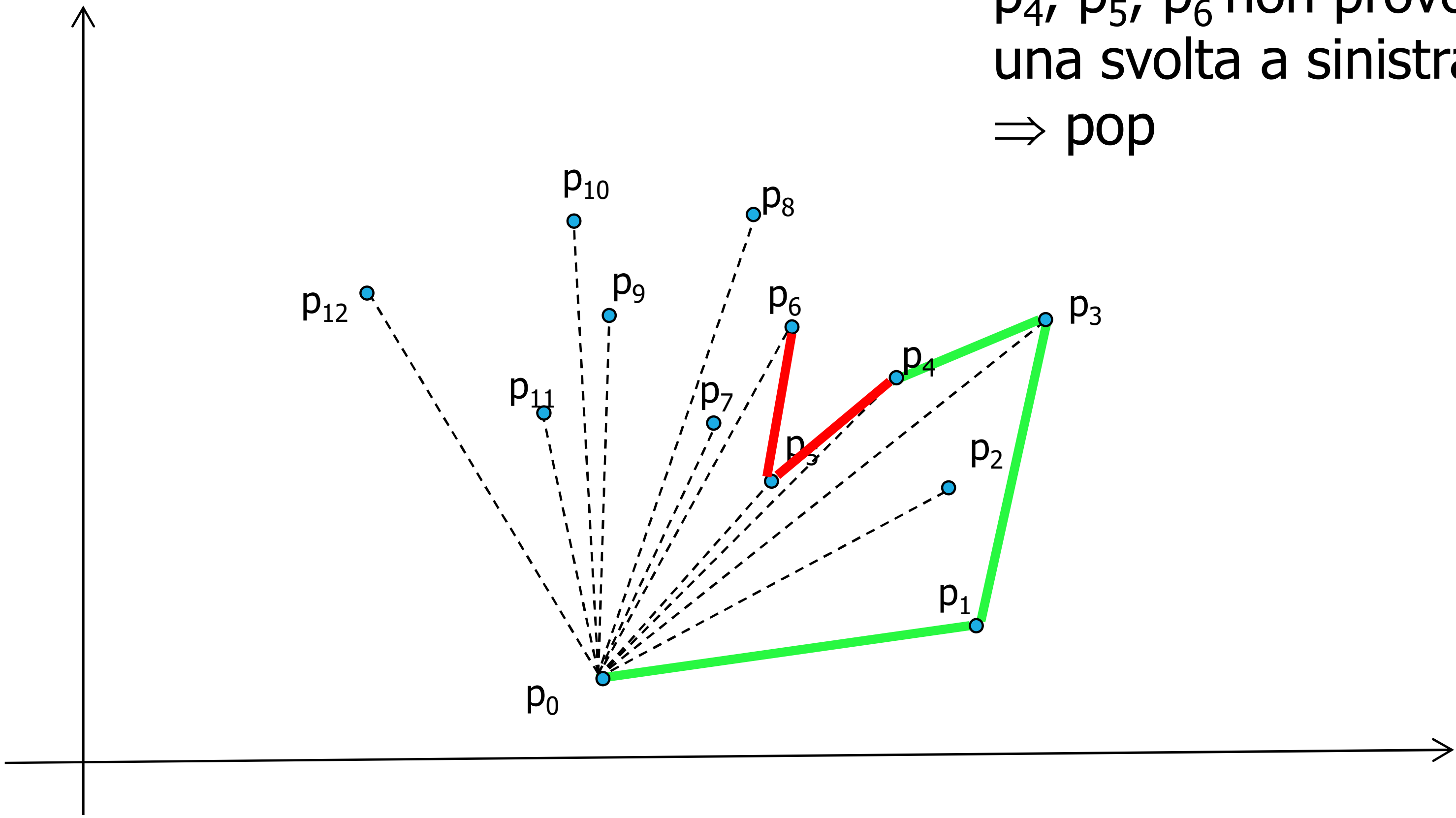
l'angolo formato da p_1, p_3, p_4 provoca una svolta a sinistra
$\Rightarrow$ push(p_4)

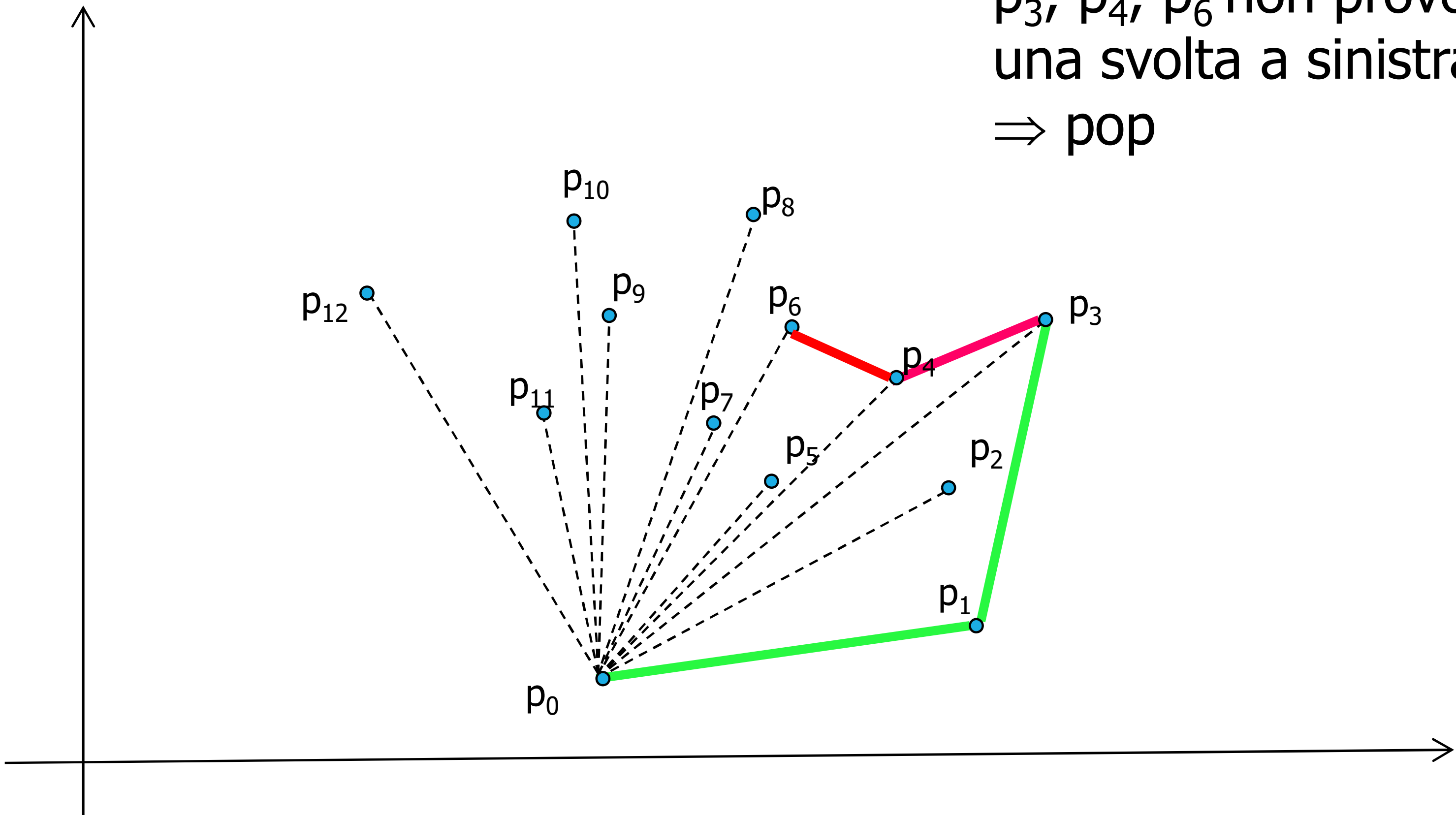


l'angolo formato da p_3, p_4, p_5 provoca una svolta a sinistra
$\Rightarrow$ push(p_5)



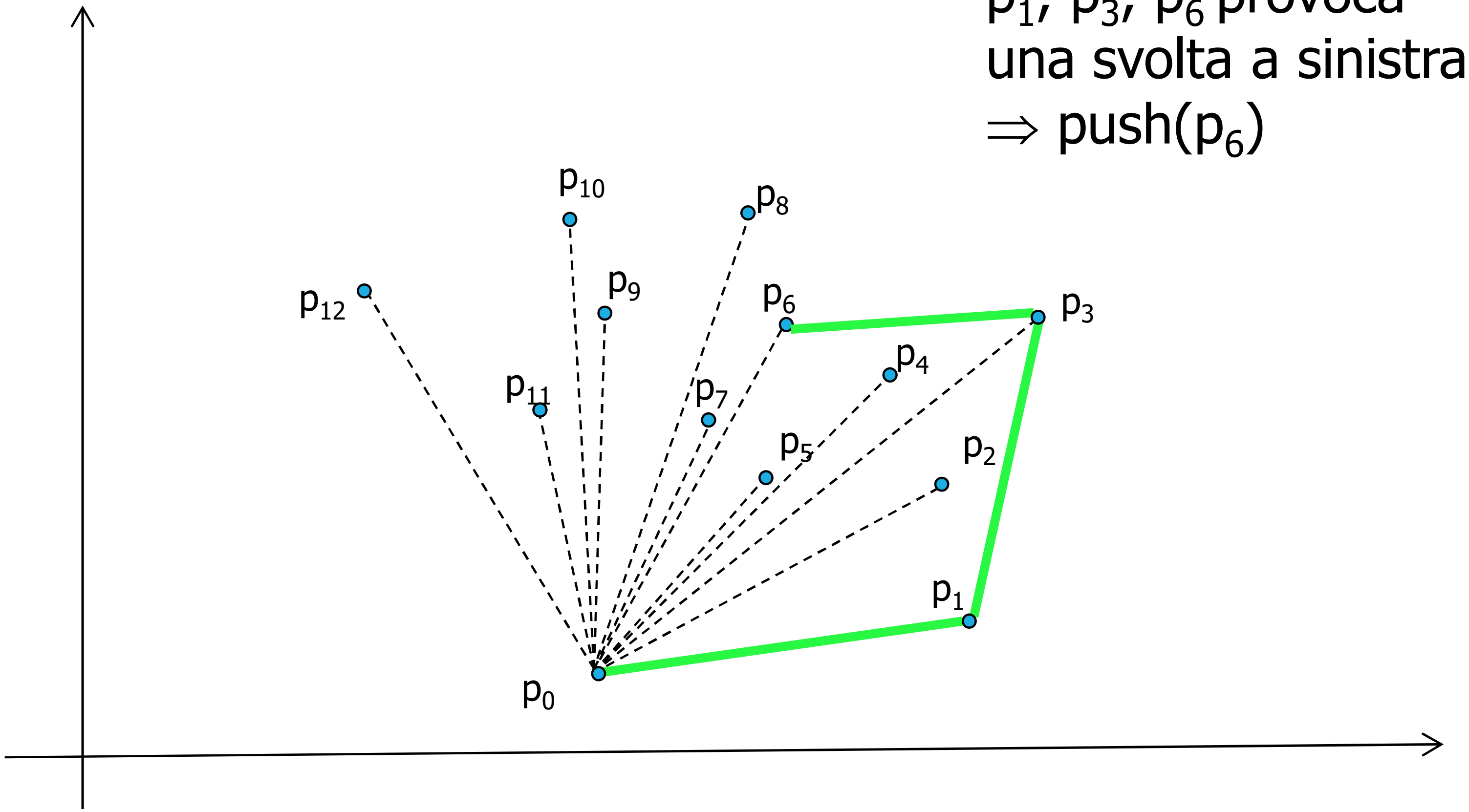
l'angolo formato da p_4, p_5, p_6 non provoca una svolta a sinistra
$\Rightarrow$ pop

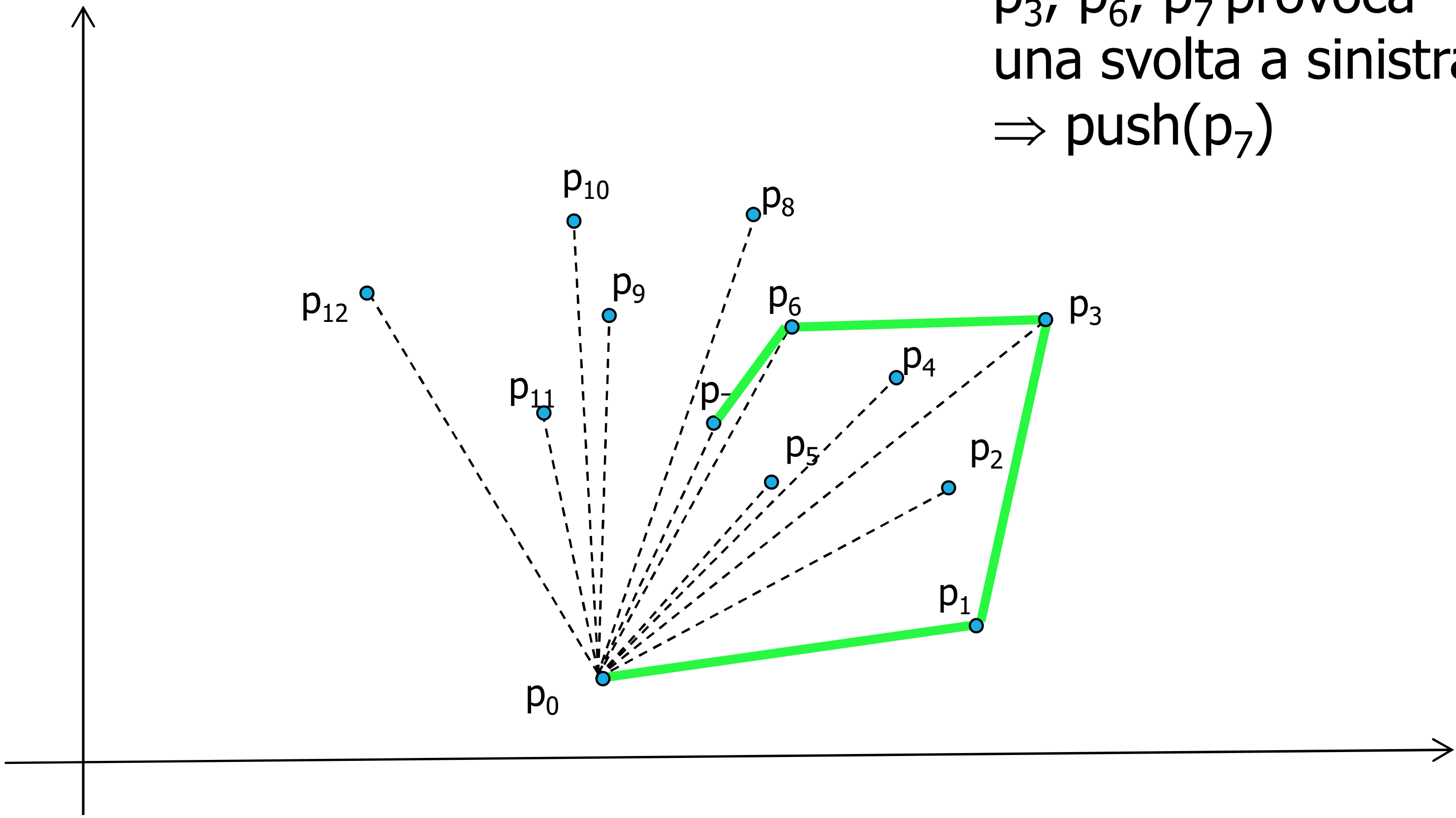




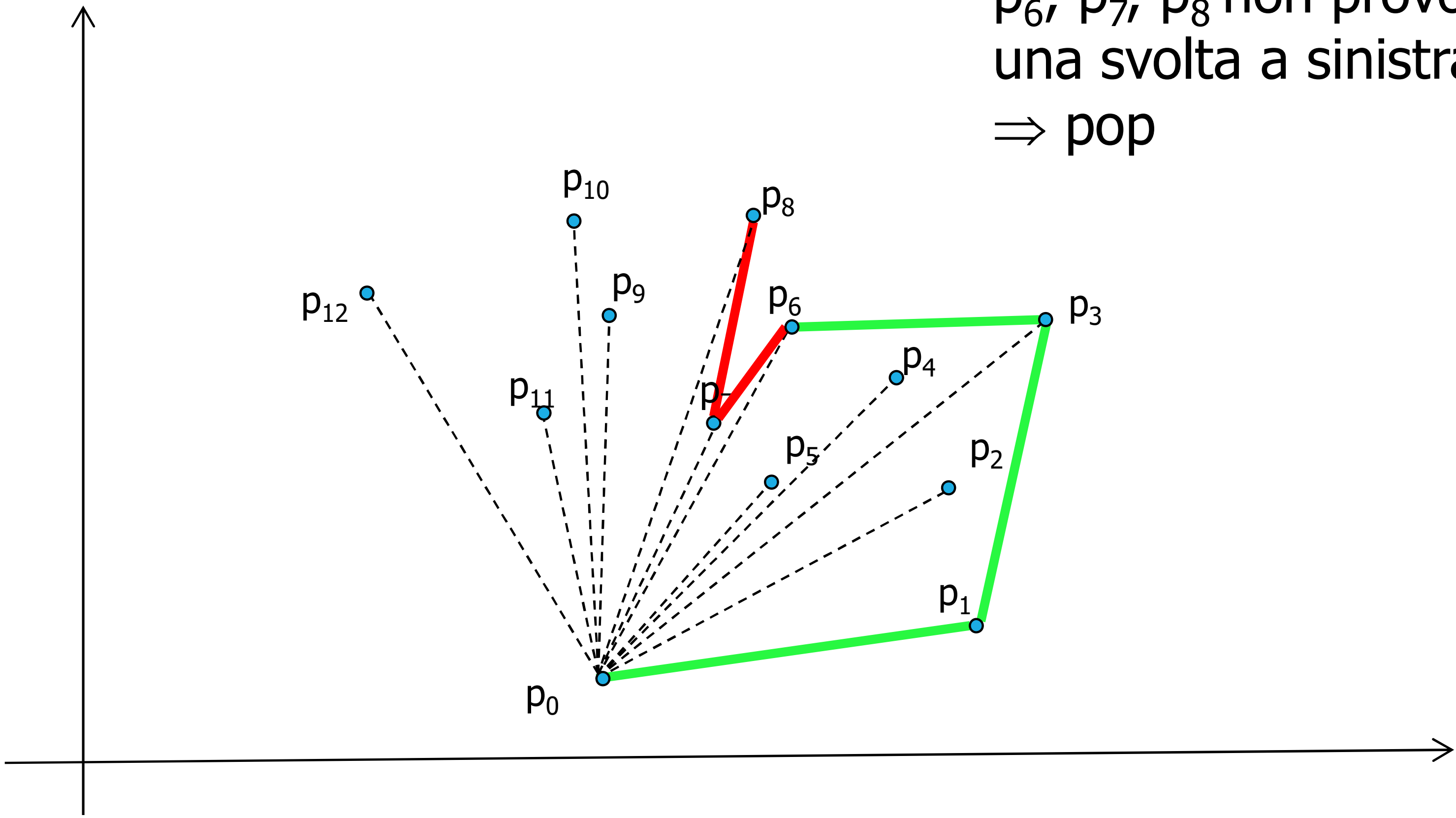
l’angolo formato da p_3, p_4, p_6 non provoca una svolta a sinistra $\Rightarrow$ pop

l'angolo formato da p_1, p_3, p_6 provoca una svolta a sinistra $\Rightarrow$ push(p_6)



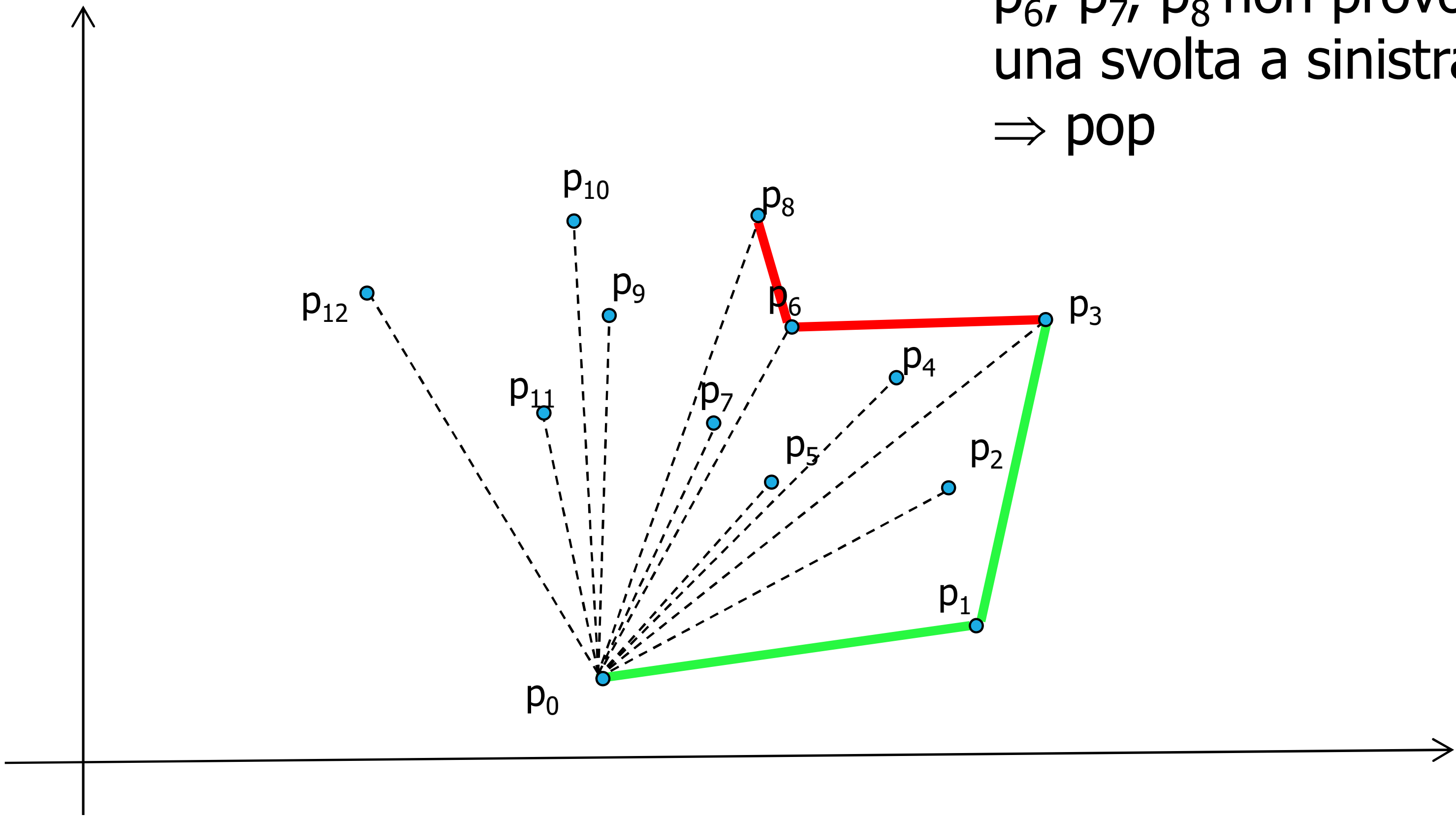


l'angolo formato da p_3, p_6, p_7 provoca una svolta a sinistra $\Rightarrow$ push(p_7)

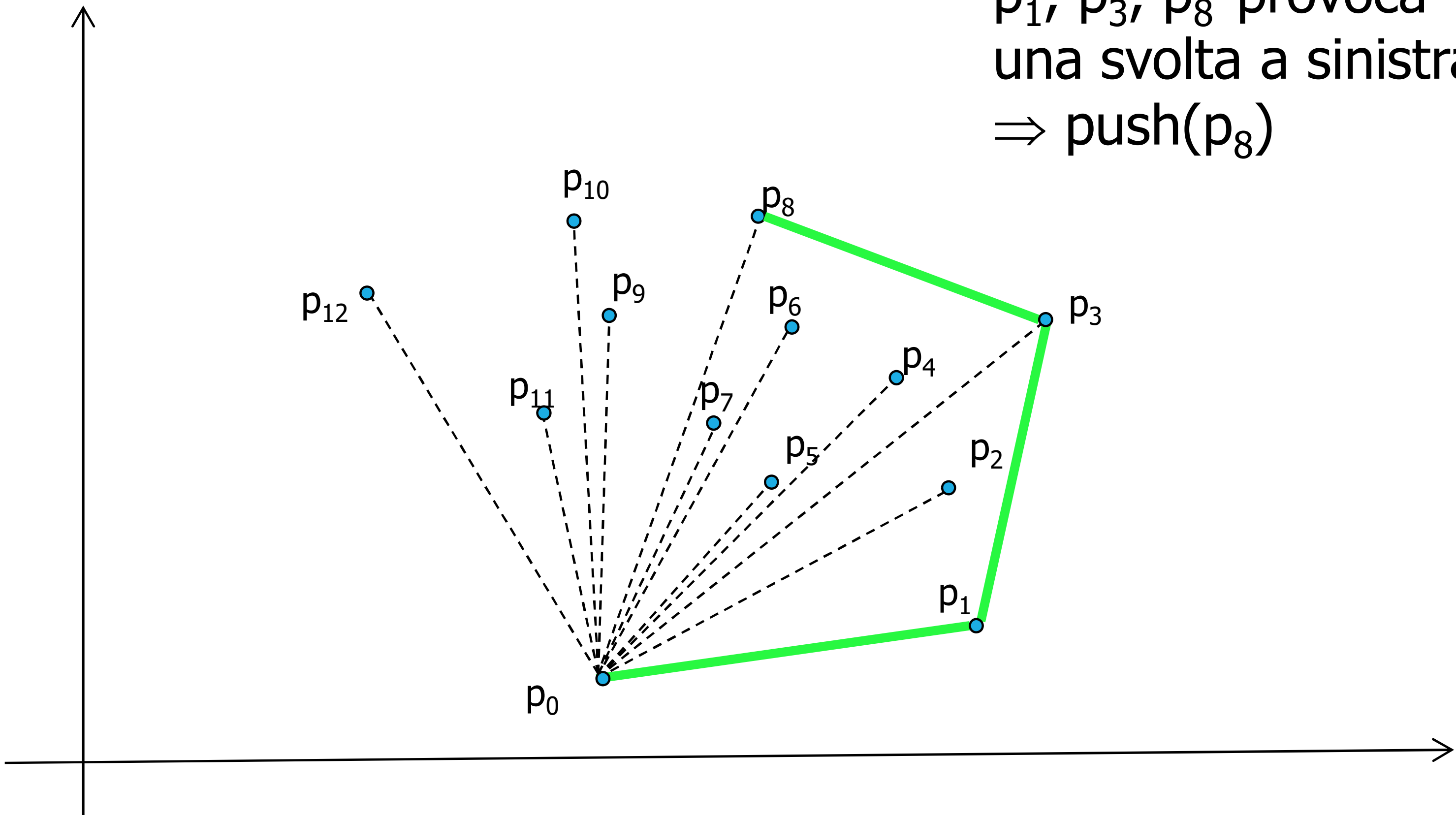


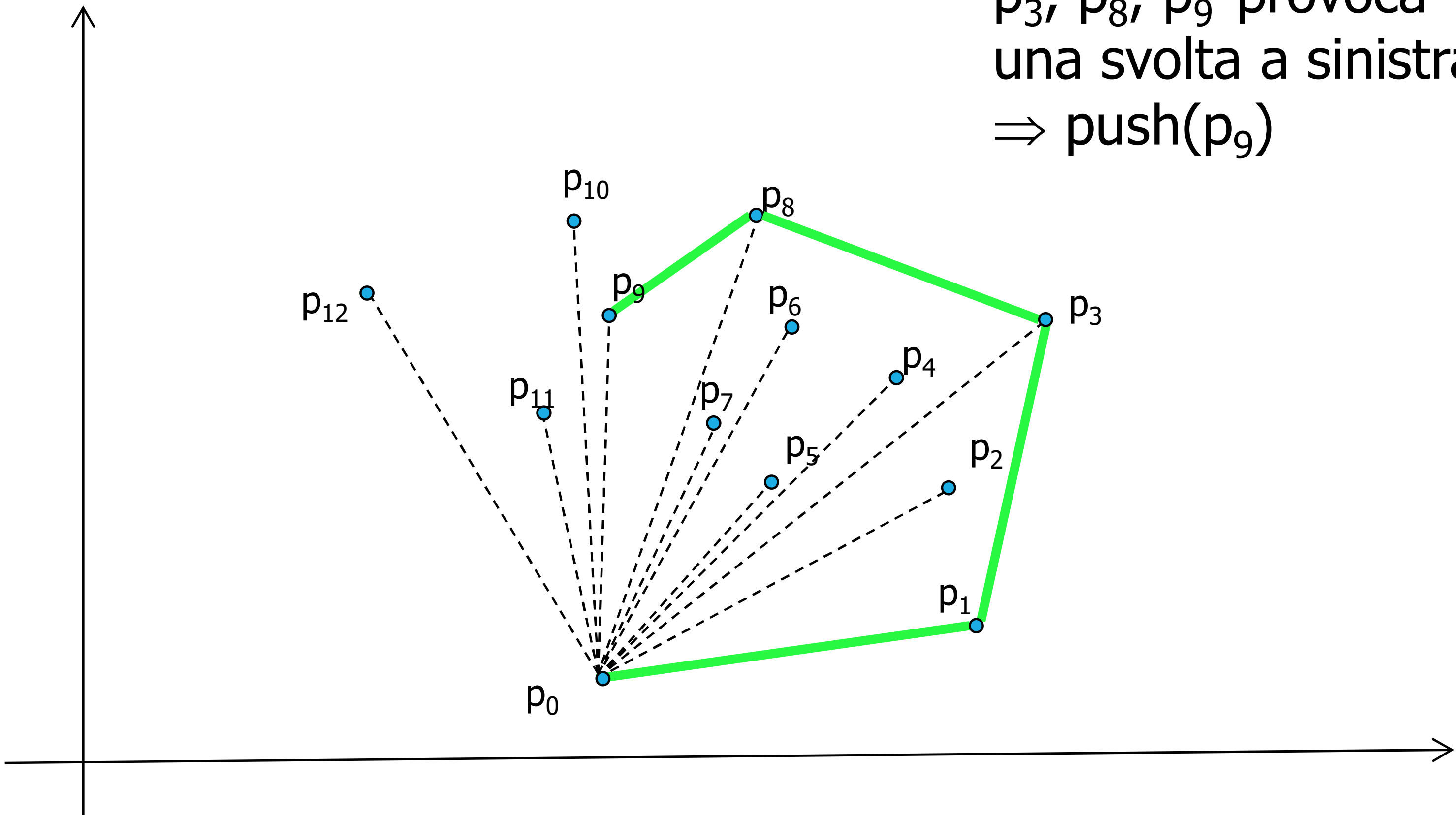
l’angolo formato da p_6, p_7, p_8 non provoca una svolta a sinistra
$\Rightarrow$ pop

l'angolo formato da p_6, p_7, p_8 non provoca una svolta a sinistra
$\Rightarrow$ pop



l’angolo formato da p_1, p_3, p_8 provoca una svolta a sinistra $\Rightarrow$ push(p_8)

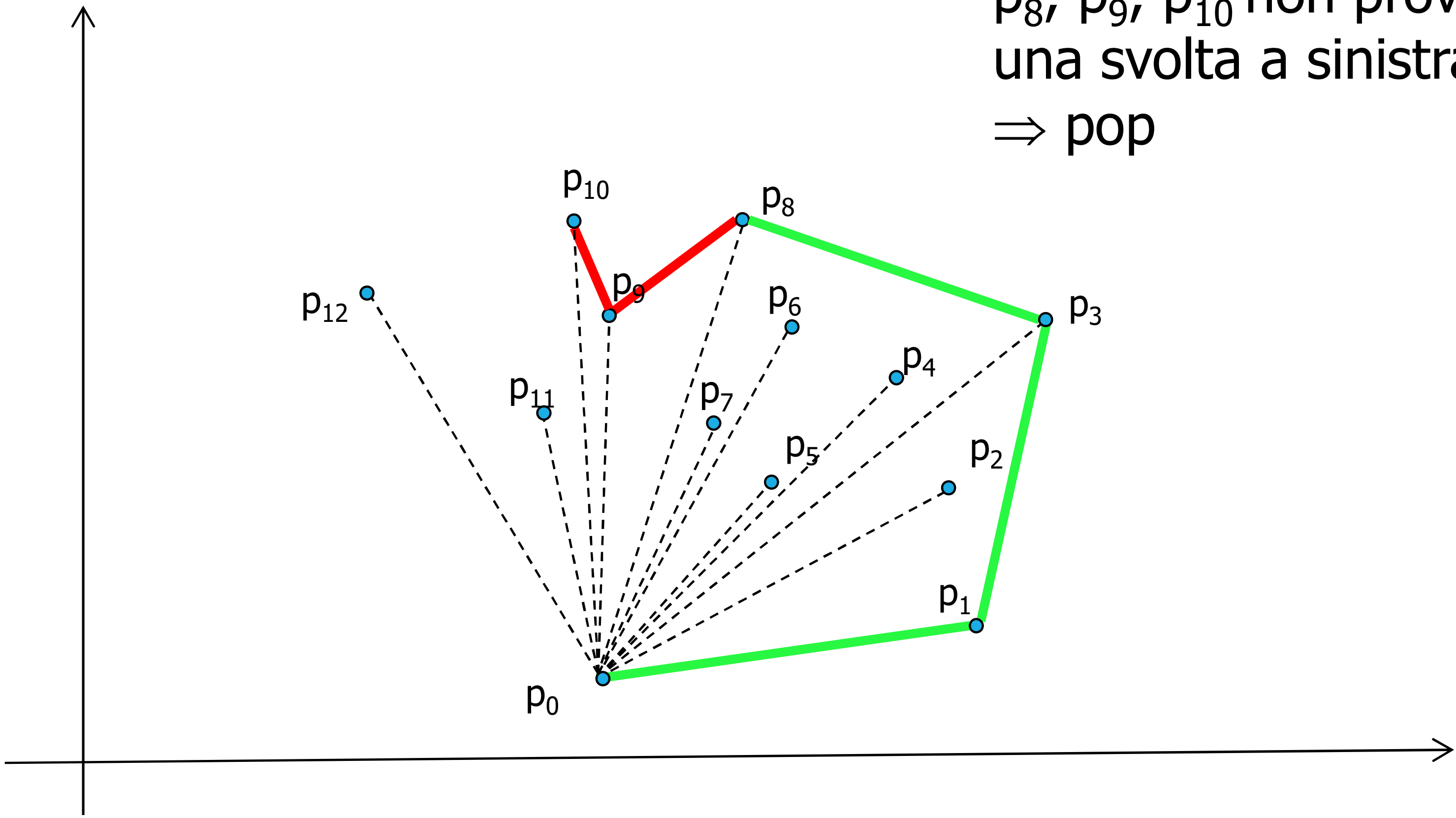




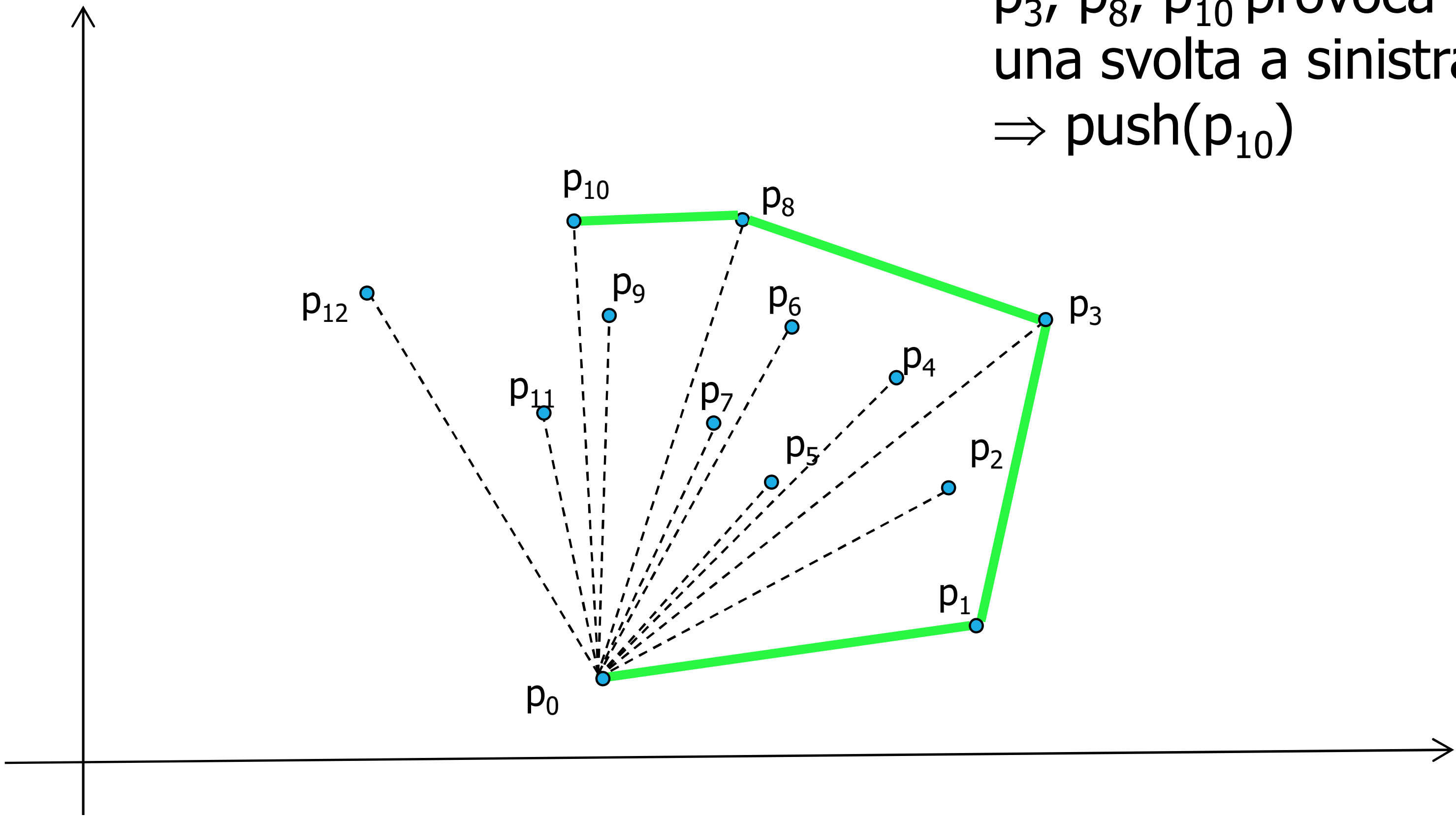
l'angolo formato da p_3, p_8, p_9 provoca una svolta a sinistra $\Rightarrow$ push(p_9)

l'angolo formato da p_8, p_9, p_{10} non provoca una svolta a sinistra

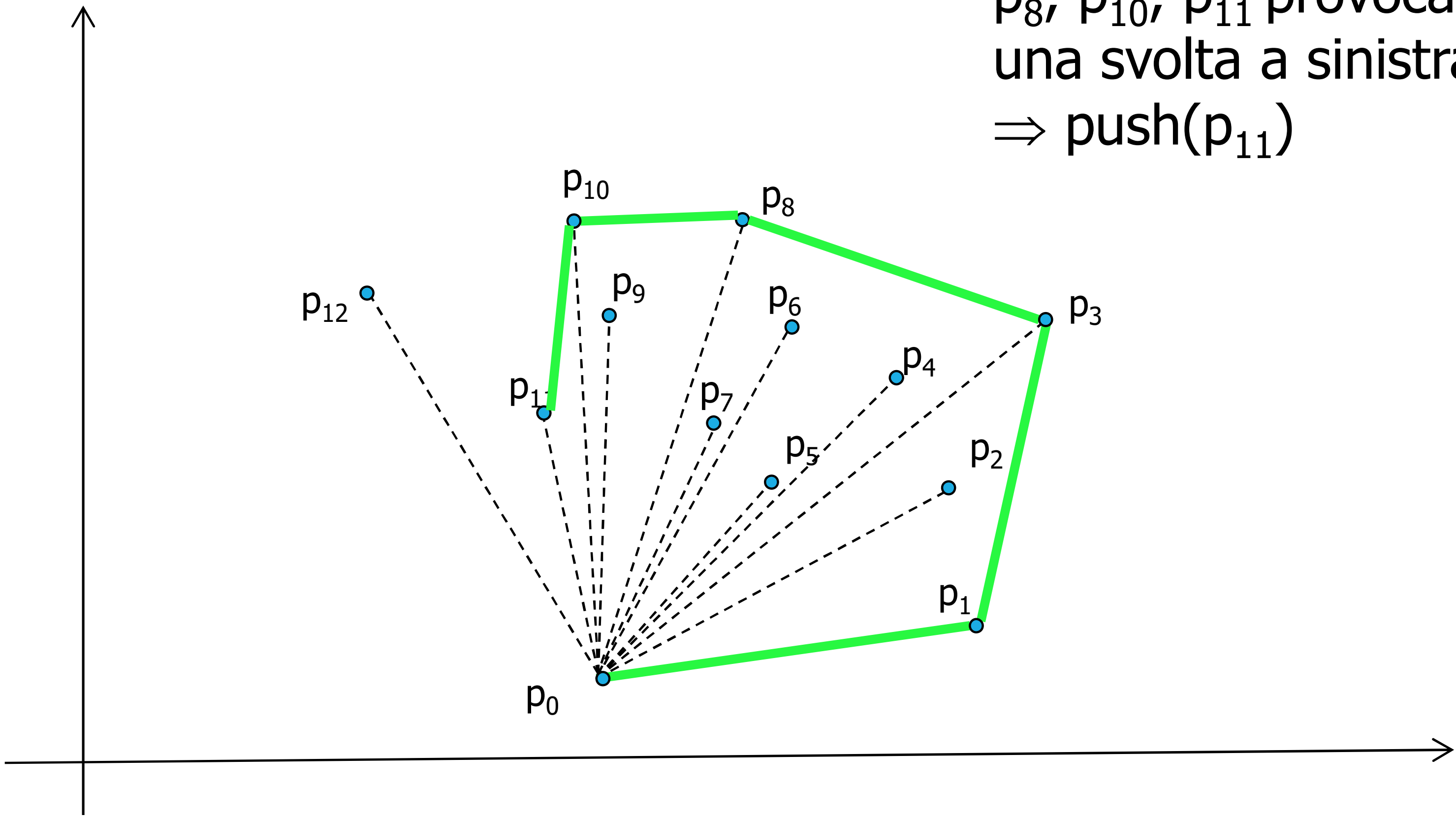
$\Rightarrow$ pop



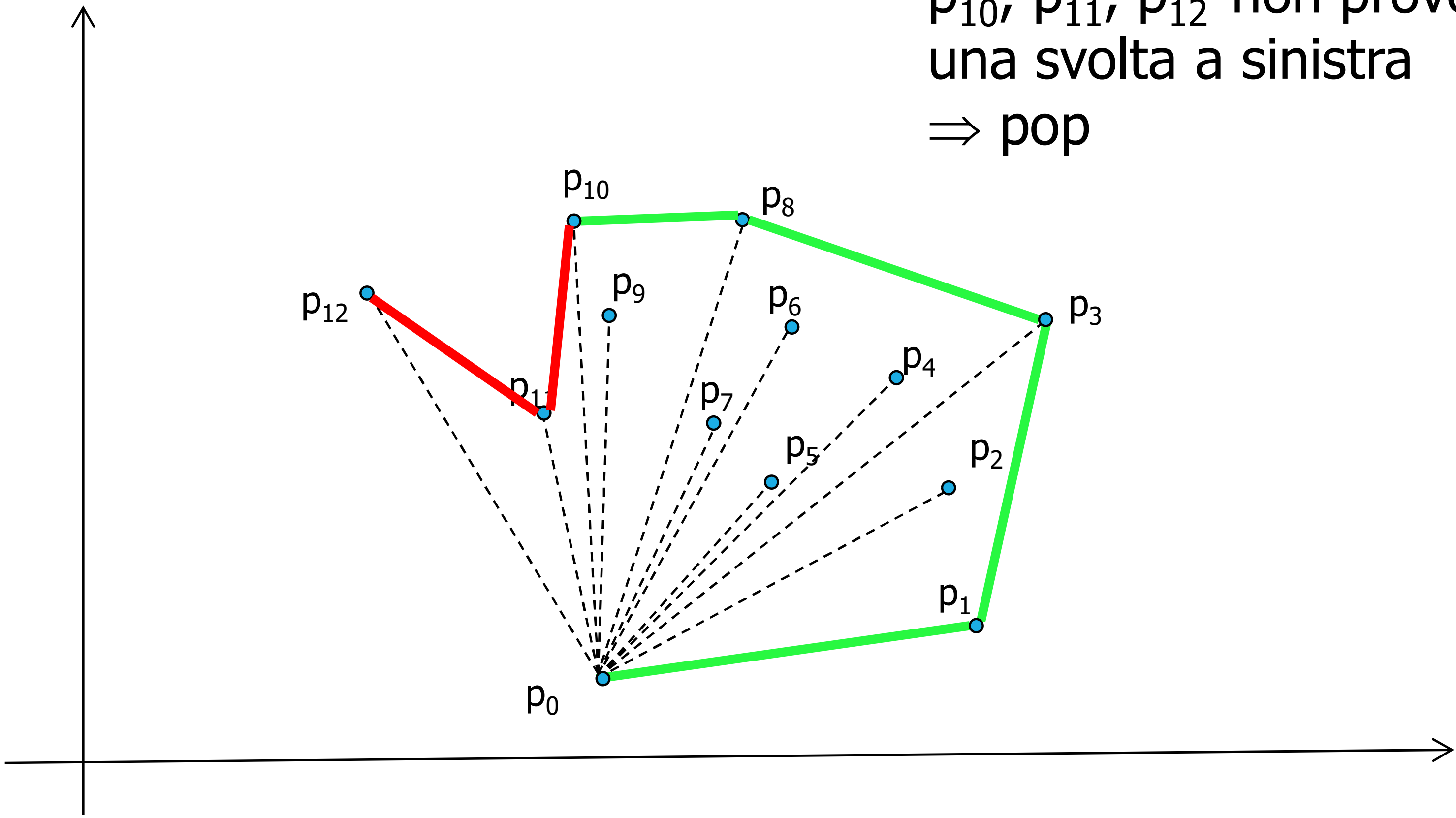
l'angolo formato da p_3, p_8, p_{10} provoca una svolta a sinistra $\Rightarrow$ push(p_{10})

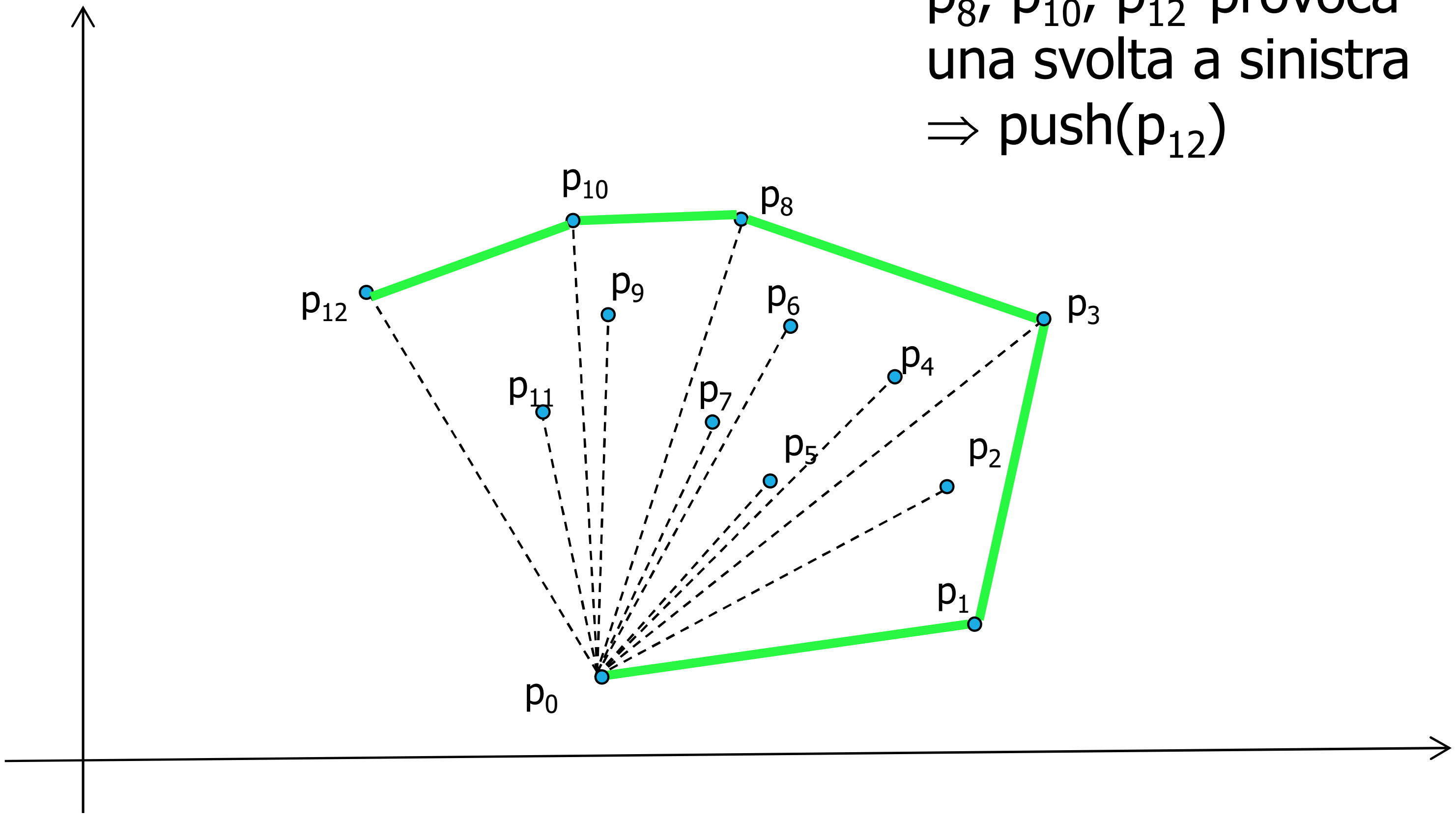


l'angolo formato da p_8, p_{10}, p_{11} provoca una svolta a sinistra $\Rightarrow$ push(p_{11})

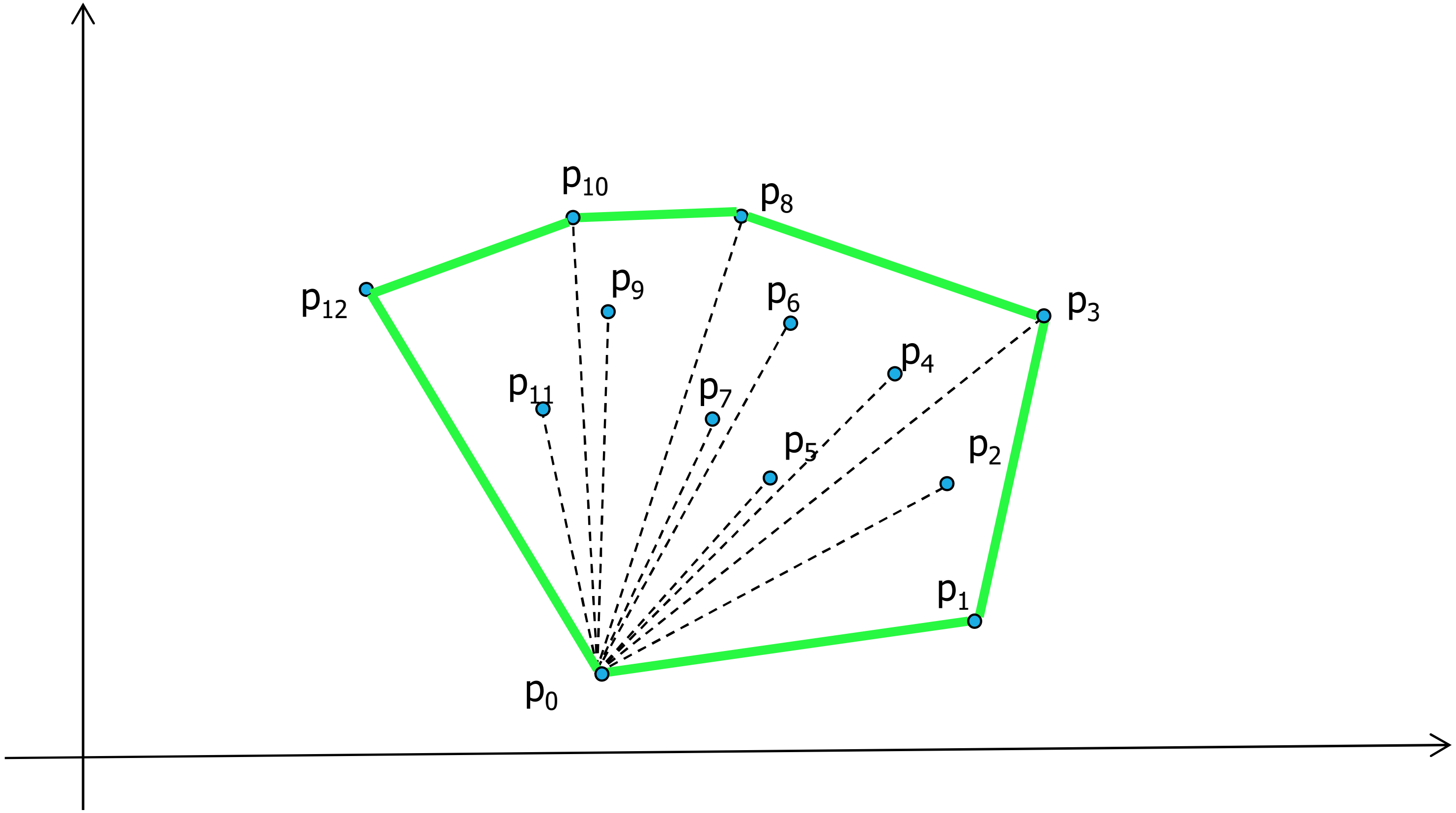


l'angolo formato da
p_{10}, p_{11}, p_{12} non provoca una svolta a sinistra
$\Rightarrow$ pop





l’angolo formato da p_8, p_{10}, p_{12} provoca una svolta a sinistra
$\Rightarrow$ push(p_{12})

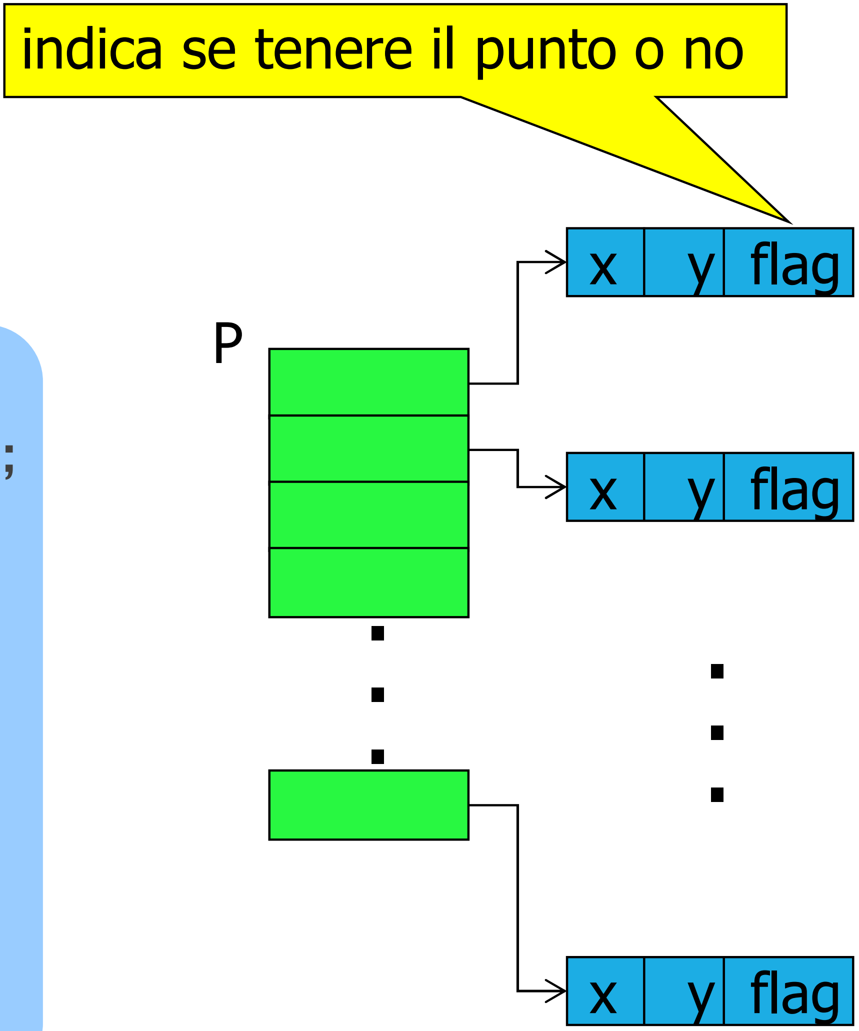

p10
p8
p12
p9
p6
p3
p4
p11
p7
p5
p2
p1
p0

Strutture dati

Vettore di puntatori a punti:

```
typedef struct point *Point;
struct point {int x, y, flag;};
Point *P;

Point NEWpoint(int x, int y){
  Point p;
  p = malloc(sizeof(*p));
  p->x = x;
  p->y = y;
  p->flag = 1;
  return p;
}
```



Identificazione del punto a ordinata minima (il più a SX in caso di parità) e memorizzazione come p_0

```
void LookForP0(void) {
  int i, min = 0;
  Point tmp;
  for(i = 1; i < N; i++) {
    if(P[i]->y < P[min]->y)
      min = i;
    else if(P[i]->y == P[min]->y)
      if(P[i]->x < P[min]->x)
        min = i;
    }
    tmp = P[0]; P[0] = P[min]; P[min] = tmp;
    return;
}
```

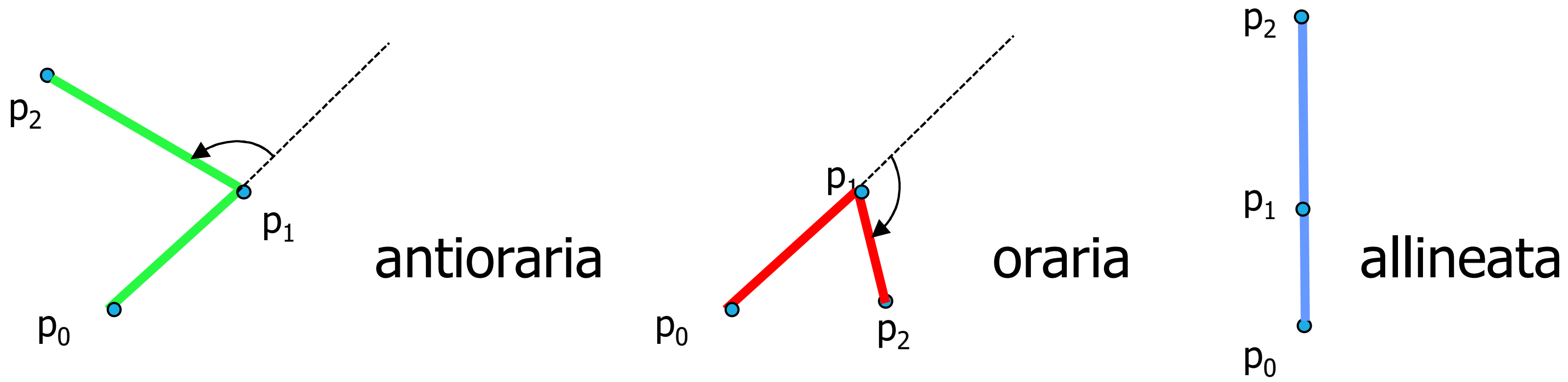
Ordinamento dei punti rimanenti $p_1 \ldots p_N$ per angolo polare crescente rispetto a p_0. Per punti sulla stessa retta, tenere solo il più distante da p_0

ordinamento con MergeSort, confronto in base all'angolo polare, senza funzioni trigonometriche costose (atan) determinazione della svolta a sinistra o a destra

per i rimanenti punti con i = 3 a N

- fintanto che l'angolo formato dal punto sotto la cima dello stack, da quello in cima allo stack e da p_i non provoca una svolta antioraria (a sinistra), pop dallo stack
- push sullo stack di p_i

Determinazione della svolta



$p_0 = (x_0, y_0)$ $p_1 = (x_1, y_1)$ $p_2 = (x_2, y_2)$

$$M = \begin{pmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

Se $\det(M) = (x_1-x_0)\cdot(y_2-y_0) - (y1-y0)\cdot(x_2-x_0)$

- >0: svolta antioraria
- <0: svolta oraria
- =0: allineamento

```c
int XProd(Point p0, Point p1, Point p2){
  int x1, x2, y1, y2, prod;
  x1 = p1->x - p0->x;
  x2 = p2->x - p0->x;
  y1 = p1->y - p0->y;
  y2 = p2->y - p0->y;
  prod = (x1 * y2) - (x2 * y1);

  if(prod > 0)
    return 1;
  else if(prod < 0)
    return -1;
  return 0;
}
```

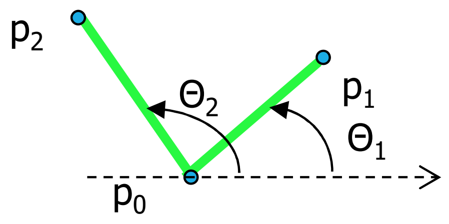
svolta antioraria

svolta oraria

allineamento

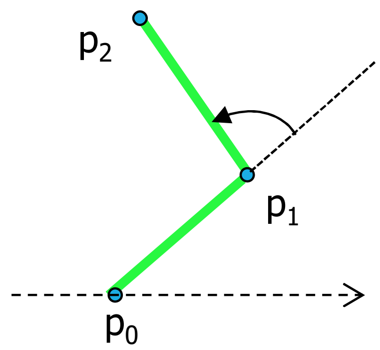
Confronto tra punti per angolo polare

$p_0 = (x_0, y_0) \quad p_1 = (x_1, y_1) \quad p_2 = (x_2, y_2)$



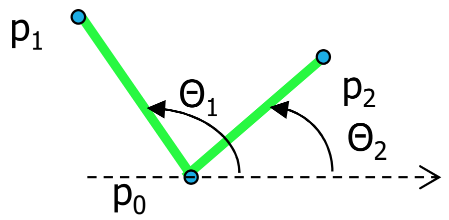
se p_0-p_1-p_2 è una svolta antioraria

`CrossProduct(P[0],p1,p2)` è 1



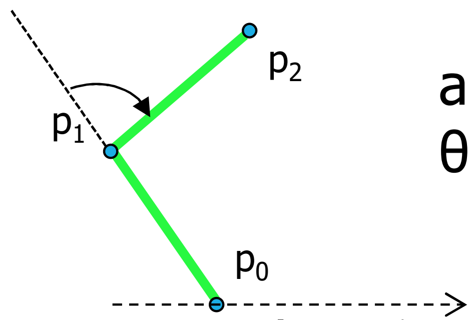
allora $\theta_2 > \theta_1$

`ComparePoints(p1,p2)` è -1



se p_0-p_1-p_2 è una svolta oraria

`CrossProduct(P[0],p1,p2)` è -1



allora $\theta_1 > \theta_2$

`ComparePoints(p1,p2)` è 1

```c
int CmpPts(Point p1, Point p2){
  int x = XProd(P[0], p1, p2);
  if(x == 0) {
    if(MoreDist(p1, p2)) {
      p2->flag = 0;
      return 1;
    }
    else {
      p1->flag = 0;
      return -1;
    }
  }
  return -x;
}
```

Punti allineati: elimina quello più vicino a P[0].

```
void GrahanScan(void) {
  int i, j;
  LookForP0(); MergeSort(1, N-1);
  STACKinit(N); STACKpush(P[0]);
  for(i = 1, j = 1; i < 3;)
    if(P[j]->flag) { i++; STACKpush(P[j++]); } else j++;
  for(i = j; i < N; i++){
    if(P[i]->flag) {
      while(XProd(STACKnext_to_top(),STACKtop(),STACKpop();P[i])!=1)
      STACKpush(P[i]);
    }
  }
  return;
}
```

complessità T(N) = O(N lg N) + O(N) = O(N lg N)

MergeSort

Scan