プログラミング応用 第2回

河瀬 康志

2016 年 6 月 20 日

mi エディタの設定

大学の備え付け PC を使う場合は mi エディタをおすすめします.

- 「mi → モード設定 → 表示」からコード文字の表示を全てオン
  - インデントの空白数がわかりやすくなります

授業スケジュール

	日程	内容
第 1 回	6/13	ガイダンス・復習
第 2 回	6/20	文字列操作（文字列整形，パターンマッチ，正規表現） 平面幾何（線分の交差判定，点と直線の距離，凸包）
第 3 回	6/27	乱数（一様分布，正規分布への変換，乱数生成） 統計（データ処理，フィッティング）
第 4 回	7/4	計算量（オーダー表記） スタックとキュー（幅優先探索，深さ優先探索）
第 5 回	7/11	二分木（ヒープ，二分探索木） ソートアルゴリズム
第 6 回	7/18	バックトラック（N クイーン問題，数独） 動的計画法（ナップサック問題）
第 7 回	7/25	最短経路探索（Warshall-Floyd, Bellman-Ford, Dijkstra） TSP（全探索，ビット DP, ヒューリスティックス）
期末試験	???	????

アウトライン

1. 文字列操作
2. 平面幾何
3. 演習

文字列の基本

```
>>> s='hoge'
>>> t="fuga"
>>> u='''hoge
... fuga
... piyo
... ''' # 複数行の文字列
>>> u
'hoge\nfuga\npiyo\n'
>>> s+t # 連結
'hogefuga'
>>> s*3 # 繰り返し
'hogehogehoge'
>>> len(s) # 長さ
4
>>> int('42') # 整数に変換
42
>>> str(42) # 文字列に変換
'42'
```

文字列のインデックス指定

```
>>> s='hoge' # 文字列はシングルクォートで囲んで表す
>>> s[2] # インデクシング
'g'
>>> s[-1] # インデクシング
'e'
>>> s[1:3] # スライシング
'og'
>>> s[:3] # 最初の 3 文字
'hog'
>>> s[2:] # 最初の 2 文字以外
'ge'
>>> s[::2] # 2 文字毎に取る
'hg'
>>> s[::-1] # -1 文字毎にとる (逆順)
'egoh'
```

文字列の分割

```
>>> '123 456 789'.split()  # 文字列を空白で分割
['123', '456', '789']
>>> 'abaacaaabc'.split('aa')  # 指定した文字列で分割
['ab', 'c', 'abc']
>>> 'abaacaaabc'.rsplit('aa')  # rsplit は右から分割
['ab', 'ca', 'bc']
>>> 'abaacaaabc'.split('aa',1)  # 分割数を指定
['ab', 'caaabc']
>>> 'abaacaaabc'.partition('aa')  # 3 分割
('ab', 'aa', 'caaabc')
>>> 'hoge\nfuga\npiyo'.splitlines()  # 行に分割
['hoge', 'fuga', 'piyo']
```

文字列の置換

```
>>> s='Hello world!'
>>> s.replace('Hello','Goodbye')
'Goodbye world!'
>>> s.replace('l','L',2)
'HeLLo world!'
>>> s.replace('l','') # 空文字にすれば削除もできる
'Heo word!'
>>> '   hoge fuga   '.strip() # 両端の空白を削除
'hoge fuga'
```

文字列の検索

```
>>> 'orange' in 'apple orange grape' # 含むかどうかの判定
True
>>> 'Hello world!'.startswith('hello') # 始まりの文字列か判定
False
>>> 'Hello world!'.endswith('world') # 終わりの文字列か判定
False
>>> 'abaabbab'.count('ab') # 何回出てくるか
3
>>> 'abaabbab'.find('ba') # 指定した文字列の現れる場所 (左から探索)
1
>>> 'abaabbab'.rfind('c') # 指定した文字列の現れる場所 (右から探索)
-1
```

その他文字列操作

```
>>> 'Hoge123'.upper() # 大文字に変換, lower() は小文字に変換
'HOGE123'
>>> 'Hoge123'.swapcase() # 大文字と小文字を入れ替え
'hOGE123'
>>> 'this is a pen.'.capitalize() # 最初のみ大文字に
'This is a pen.'
>>> 'Hoge123'.ljust(10) # 左寄せ.  rjust, center だと右寄せ中央寄せ
'Hoge123   '
>>> '123'.zfill(10) # 0 埋め左寄せ
'0000000123'
>>> 'abc'.isalpha() # 文字列が全てアルファベットか
True
```

書式付き出力

```
>>> '{0}, {1}, {2}, {0}'.format('a', 'b', 'c') # ポジション引数による
アクセス
'a, b, c, a'
>>> '{}, {}, {}'.format('a', 'b', 'c') # version 2.7 以降
'a, b, c'
>>> '{hour}:{min}'.format(hour='14',min='52')
'14:52'
>>> '{0:<10}'.format('123') # 10 文字幅，左寄せ
'       123'
>>> '{0:a>10}'.format('123') # 10 文字幅，右寄せ，a で埋める
'aaaaaaa123'
>>> '{:.2f}'.format(3.141592) # 小数点以下 2 桁
'3.14'
>>> '{:>10.3f}'.format(3.141592)
'     3.142'
```

正規表現の基本

- 文字列の集合を一つの文字列で表現する方法の一つ
  - 例えば'a[bc]d' という正規表現は {'abd','acd'} という集合を表す
- 指定された集合に属するかどうか（マッチするか）を判定する
- 以下の記号は特別な意味をもつ（メタ文字）

```
.   ^   $   [   ]   *   +   ?   |   (   )
```

- メタ文字を使いたいときはバックスラッシュを付けてエスケープする

```
\.   \^   \$   \[   \]   \*   \+   \?   \|   \(   \)
```

- エディタでの検索，置換などでも使える
- より詳しくは `http://docs.python.jp/2/howto/regex.html` 参照

単純なパターン

.	任意の一文字にマッチ
^	行の先頭にマッチ
$	行の終端にマッチ
|	左の文字列または右の文字列にマッチ.「x|yz」は「x」または「yz」にマッチ
[...]	括弧内に含まれる一文字にマッチ.「[abc]」は「a」か「b」か「c」にマッチする.「[a-z]」だと小文字のアルファベットにマッチ.
[^...]	括弧内に含まれない一文字にマッチ.「[^a-z]」は小文字のアルファベット以外にマッチ.
*	直前の文字の 0 回以上の繰り返しにマッチ
+	直前の文字の 1 回以上の繰り返しにマッチ
$\{m,n\}$	直前の文字の m 回以上 n 回以下の繰り返しにマッチ
?	直前の文字の 0 回か 1 回の繰り返しにマッチ
(...)	グループ化をする.「(ab)+」は「ab」の繰り返しにマッチ
\1	グループの 1 番目の中身とマッチ
\d	数字とマッチ.「[0-9]」と同じ
\s	空白文字とマッチ.「[\t\n\r\f\v]」と同じ
\w	英数文字下線とマッチ.「[a-zA-Z0-9_]」と同じ

Python での使い方

```
>>> import re # 正規表現ライブラリを読み込み
>>> m = re.search('a.*b.*c', 'abracadabra') # マッチしなければ None
>>> m.group() # マッチした文字列
'abrac'
>>> m.start() # マッチの開始位置
0
>>> m.end() # マッチの終了位置
5
>>> m.span() # マッチ位置 (start,end) のタプル
(0,5)
>>> re.findall('[^a]+a', 'abracadabra') # マッチする部分文字列を全て取り出す
['bra', 'ca', 'da', 'bra']
>>> re.split('[ab]+', 'abracadabra') # マッチする部分で分割
['', 'r', 'c', 'd', 'r', '']
>>> re.sub('[^ab]', 'x', 'abracadabra') # 置換
'abxaxaxabxa'
>>> print re.sub('(\\d+)/(\\d+)', '\\1 月\\2 日', ' 今日は 6/20 です')
' 今日は 6 月 20 日です'
```

正規表現の例

郵便番号にマッチ

```
\d{3}-?\d{4}
```

携帯・PHS 番号にマッチ

```
0[789]0-?\d{4}-?\d{4}
```

メールアドレスに (ほぼ) マッチ

```
[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*
```

バックスラッシュ使用時の注意

- Python の文字列では「\」自体にエスケープが必要
- 「\」にマッチする正規表現「\\」を表す文字列は'\\\\' となる
- 'r' を文字列リテラルの先頭に置けば解決（raw string）

```
>>> import re
>>> bool(re.search('\\\\section', '\\section'))
True
>>> bool(re.search(r'\\section', '\\section'))
True
```

応用： 形式言語の階層

- Σ: アルファベット (有限集合． $\{a, b, c\}, \{0, 1\}$ など)
- Σ^*: 文字列．アルファベット Σ を並べたもの．
- $L \subseteq \Sigma^*$: (形式) 言語．許される文字列を集めたもの．

上ほど表現力が高い

- 帰納的可算言語： チューリングマシンによって表現できる言語
- 文脈依存言語： 線形拘束オートマトン
- 文脈自由言語： プッシュダウンオートマトン
- 正規言語： 正規表現 (または有限オートマトン)

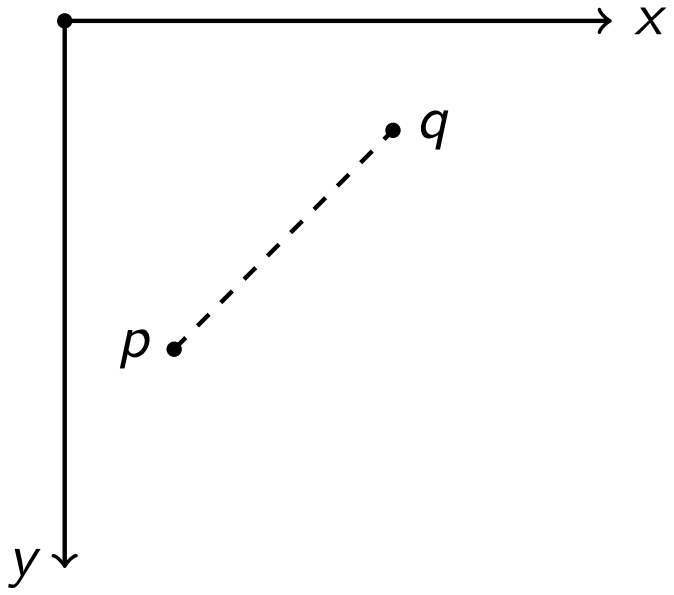
アウトライン

1. 文字列操作
2. 平面幾何
3. 演習

点の距離

点が x 座標と y 座標のペアで表されているとする

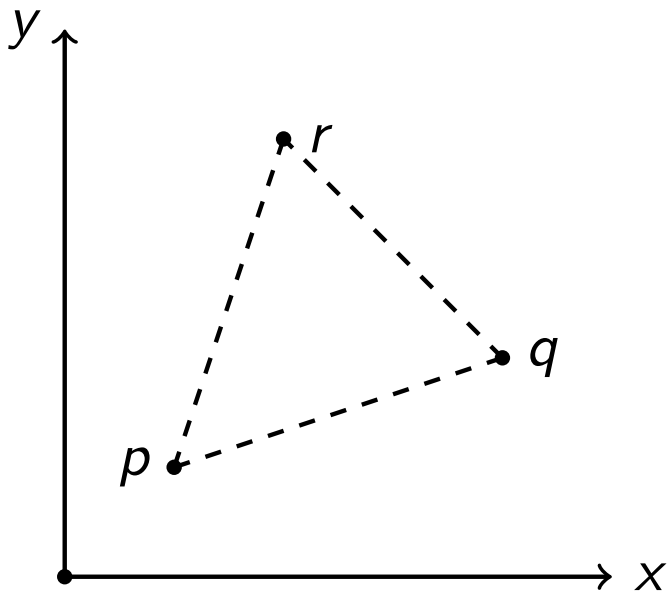
```
>>> def dist(p,q):
...     return ((p[0]-q[0])**2+(p[1]-q[1])**2)**0.5
...
>>> s=(100,40)
>>> t=(30,60)
>>> dist(s,t)
72.80109889280519
```

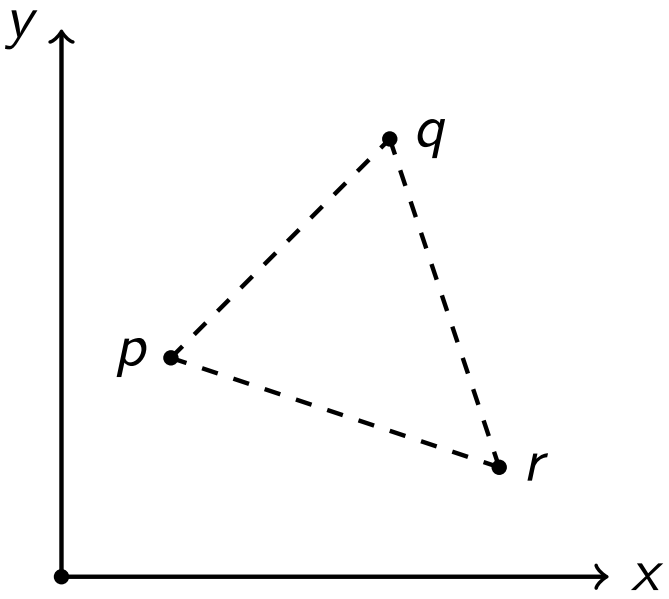


三角形の符号付き面積

点 p, q, r が反時計周りに並んでいるなら正，時計回りなら負

```
def area(p,q,r):
    return ((q[0]-p[0])*(r[1]-p[1])-(r[0]-p[0])*(q[1]-p[1]))/2.0
```

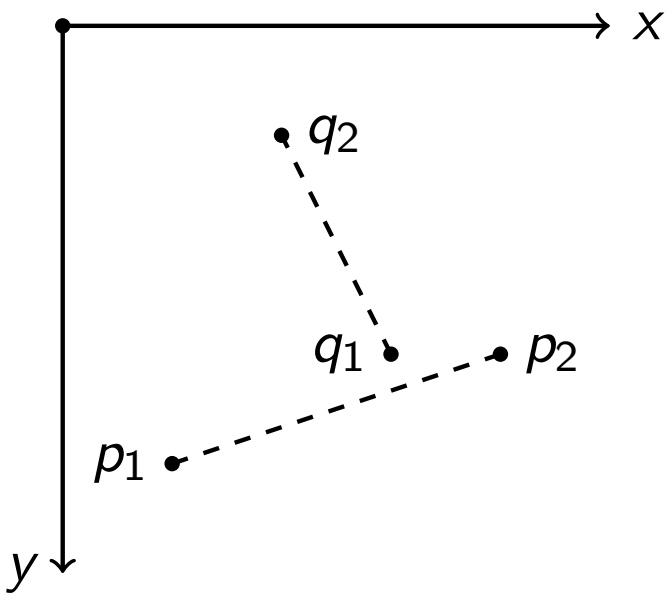


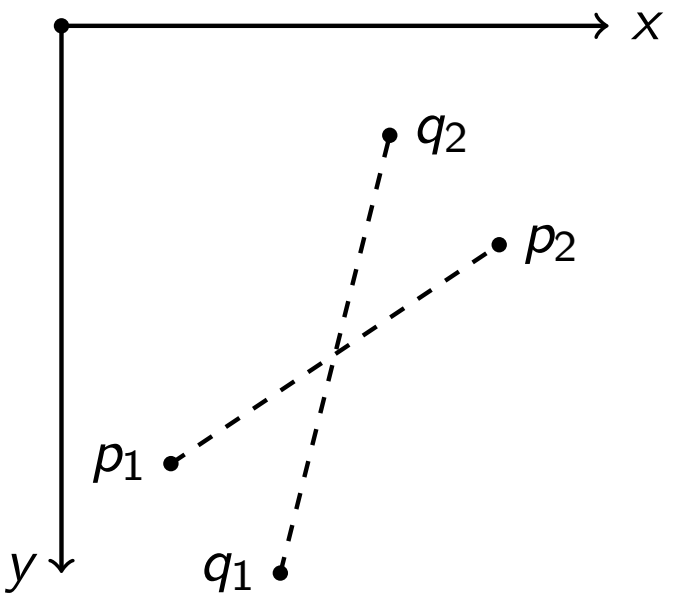


線分の交差判定

直線 p_1-p_2 に対して q_1 と q_2 が逆側かつ,
直線 q_1-q_2 に対して p_1 と p_2 が逆側なら交差

```
def intersect(p1,p2,q1,q2):
    return (area(p1,p2,q1)*area(p1,p2,q2)<0) and \
           (area(q1,q2,p1)*area(q1,q2,p2)<0)
```

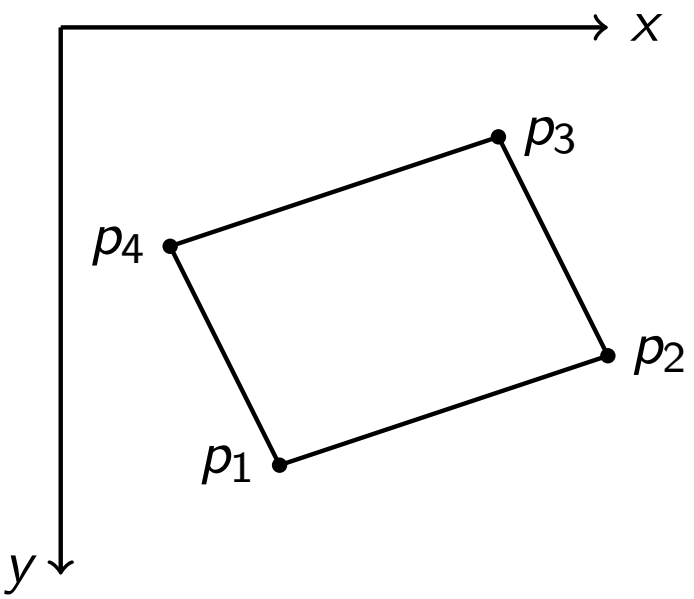


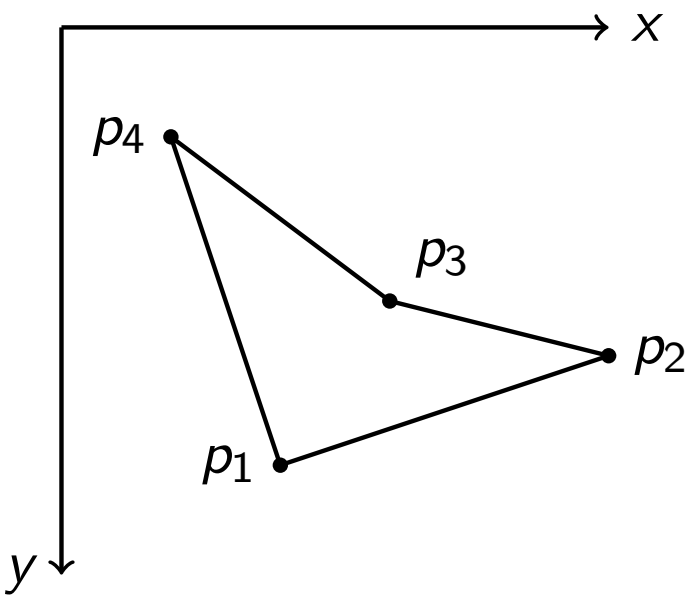


多角形が凸であるかどうか

- 多角形は座標のペアのリストで表されるとする
- p_i-p_{i+1}-p_{i+2} が全て同じ方向にあれば凸

```
def isconvex(ps):
    n = len(ps)
    a = area(ps[0],ps[1],ps[2])
    for i in range(n):
        if a*area(ps[i%n],ps[(i+1)%n],ps[(i+2)%n])<=0:
            return False
    return True
```





Tkinter

- Tkinter の canvas を用いると描画できます
  - Python 3 では tkinter と小文字に変わっていることに注意
- 詳しくは http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/canvas.html

plot.py

```
# -*- coding: utf8 -*-
import Tkinter
root = Tkinter.Tk()
canv = Tkinter.Canvas(root, width = 800, height = 600)
canv.pack()

# ここに処理を書く
#

root.mainloop()
```

図形を描く

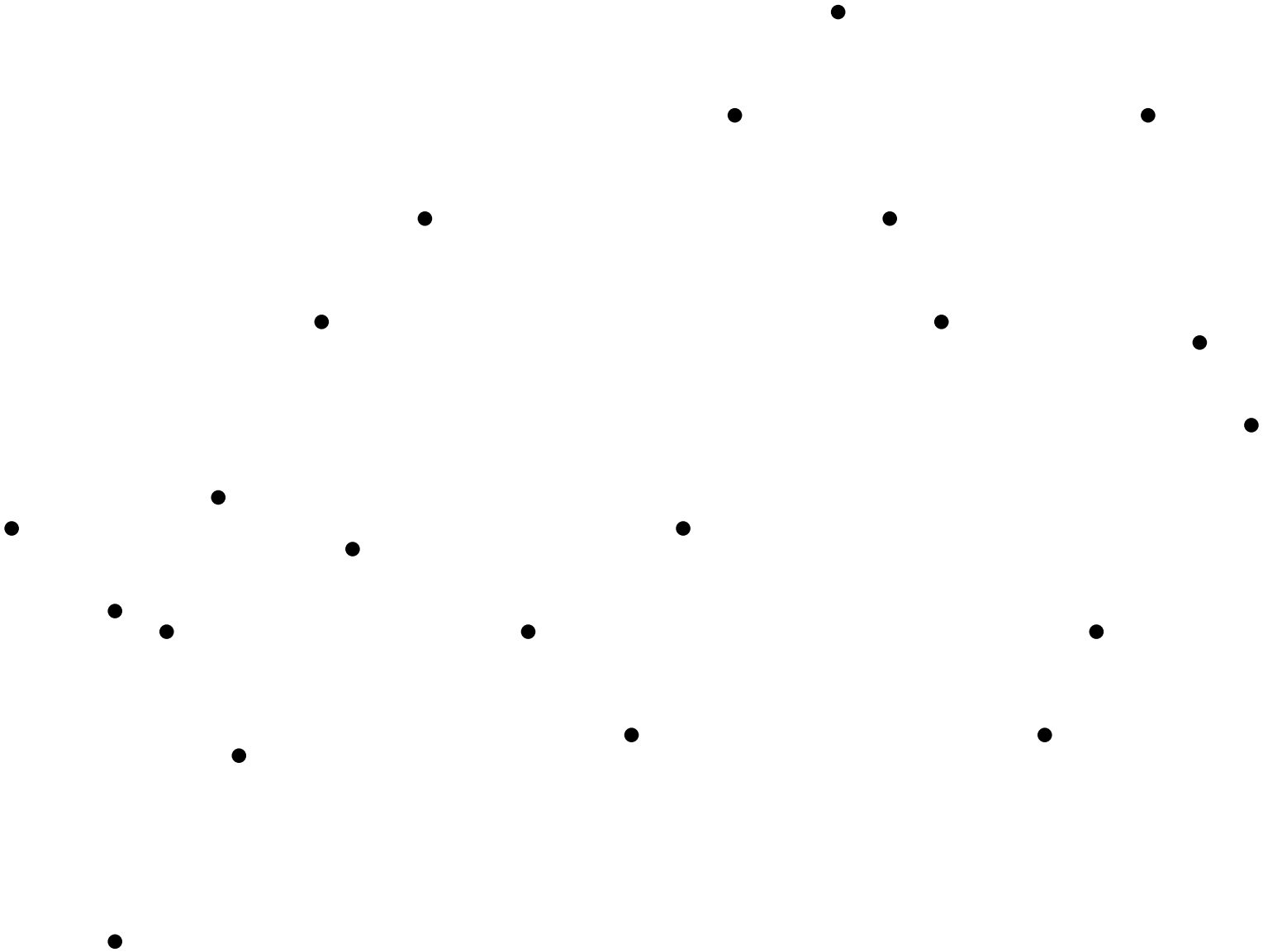
```
canv.create_line(x1,y1,x2,y2) # (x1,y1) と (x2,y2) を線で結ぶ
canv.create_line(x1,y1,x2,y2,x3,y3) # 点の数を増やして，順に線で結ぶ
canv.create_rectangle(x1,y1,x2,y2) # 長方形
canv.create_oval(x1,y1,x2,y2) # 楕円
canv.create_polygon(x1,y1,x2,y2,x3,y3,x4,y4) # 多角形
canv.create_text(x,y,text='hello') # (x,y) に'hello' を表示

# fill = 色 を付けると塗りつぶし色を変えられます
# outline = 色 を付けると縁の色を変えられます
# width = 幅 を付けると縁の太さを変えられます
canv.create_rectangle(100,100,200,200,fill='red')
canv.create_oval(100,100,200,200,outline='#c0ffee',width=4.0)
```

- 色は'#rrggbb' で指定可能 (16 進数)
  - '#ffffff' は白
  - '#000000' は黒
  - '#00ffff' はシアン

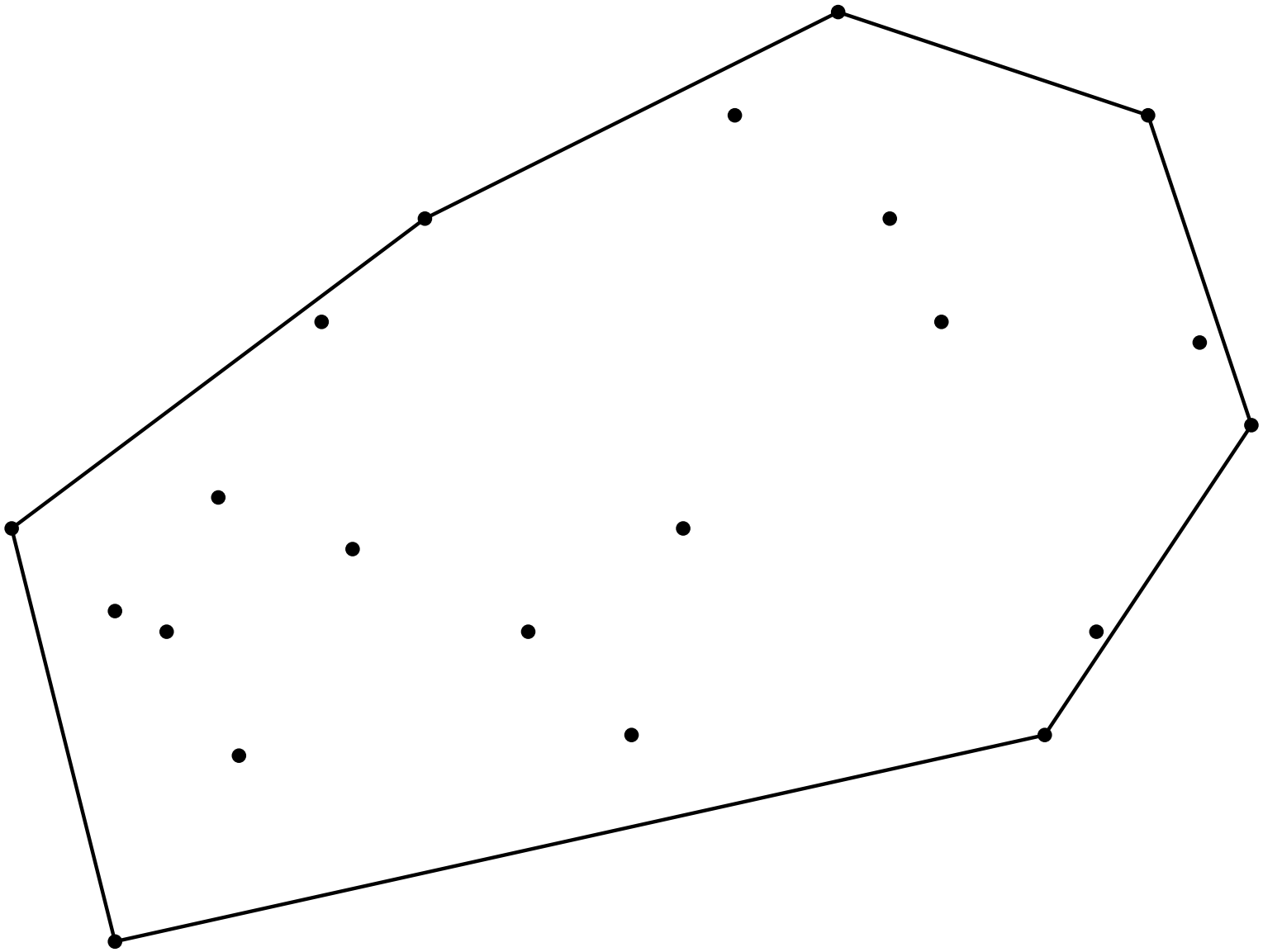
ランダムな点配置

```
import Tkinter
import random
w,h,m = 800,600,50
root = Tkinter.Tk()
canv = Tkinter.Canvas(root, width = w, height = h)
canv.pack()

n=100
ps = [(random.uniform(m,w-m),random.uniform(m,h-m)) for i in range(n)]
for p in ps:
    canv.create_oval(p[0], p[1], p[0], p[1],width='10')

root.mainloop()
```

凸包

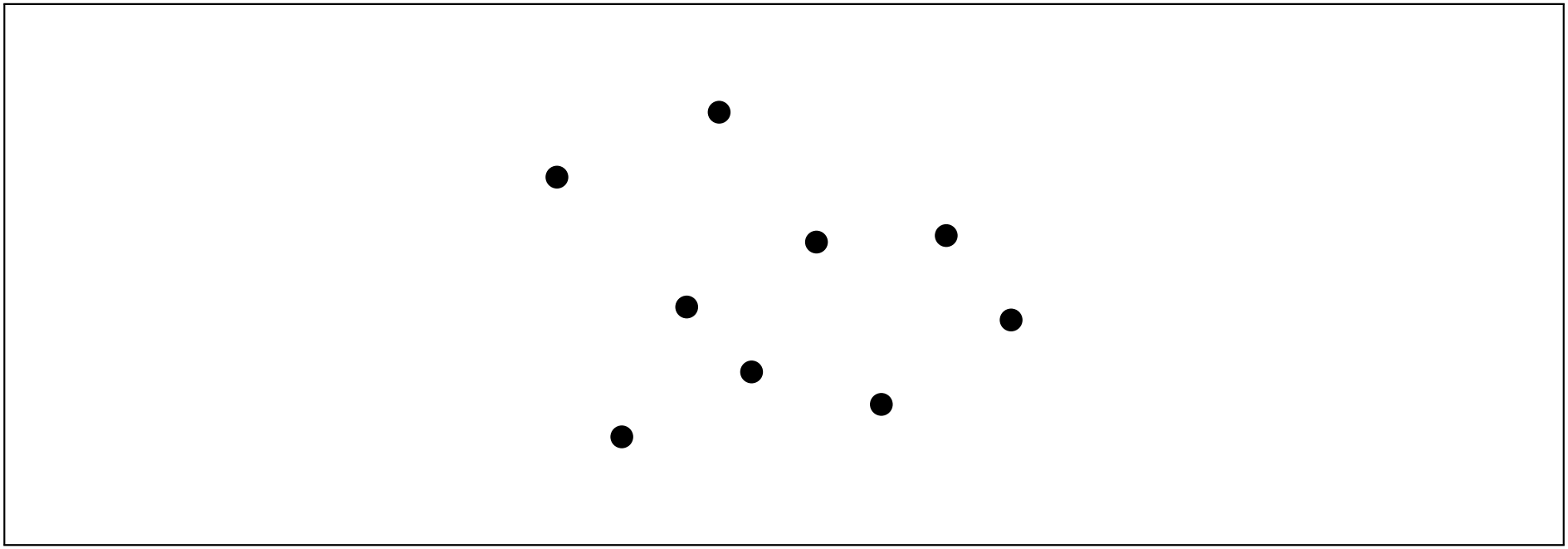


凸包



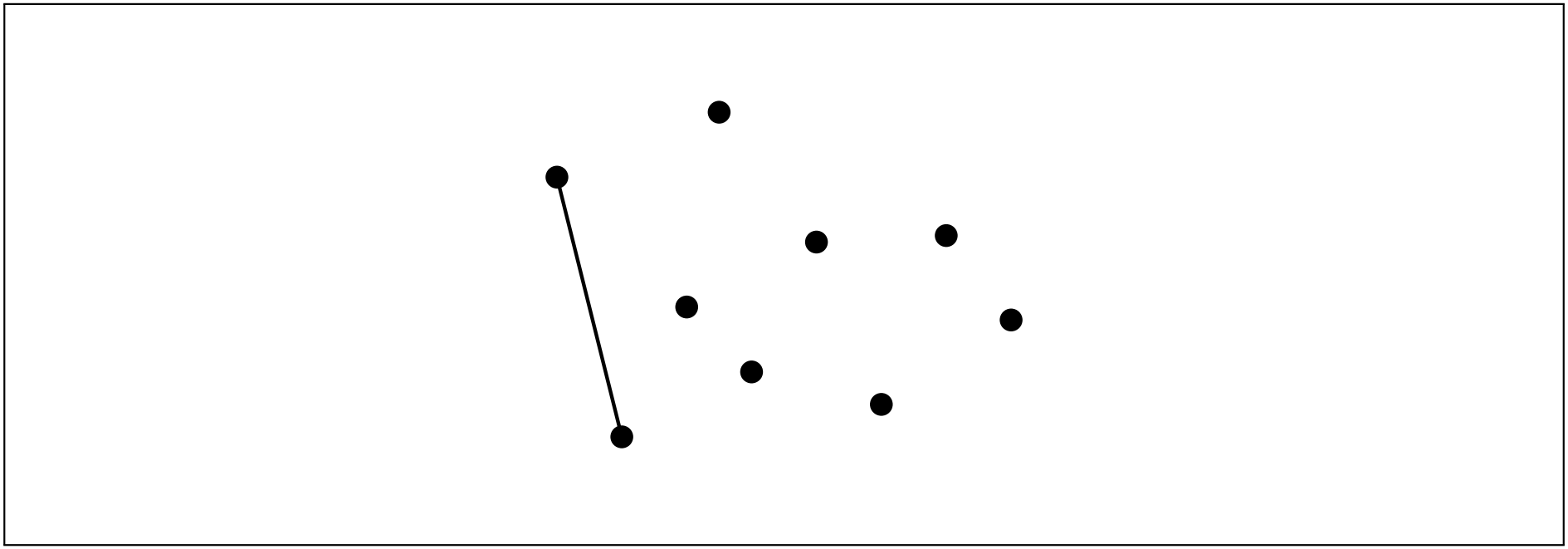
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



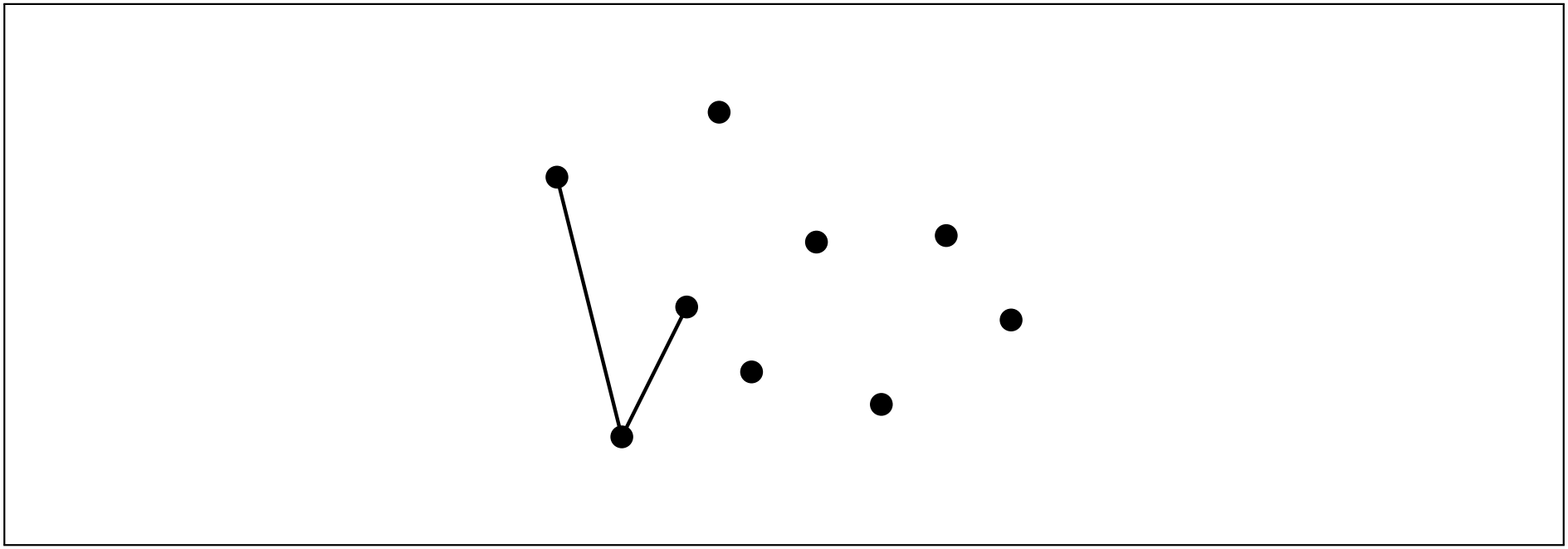
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



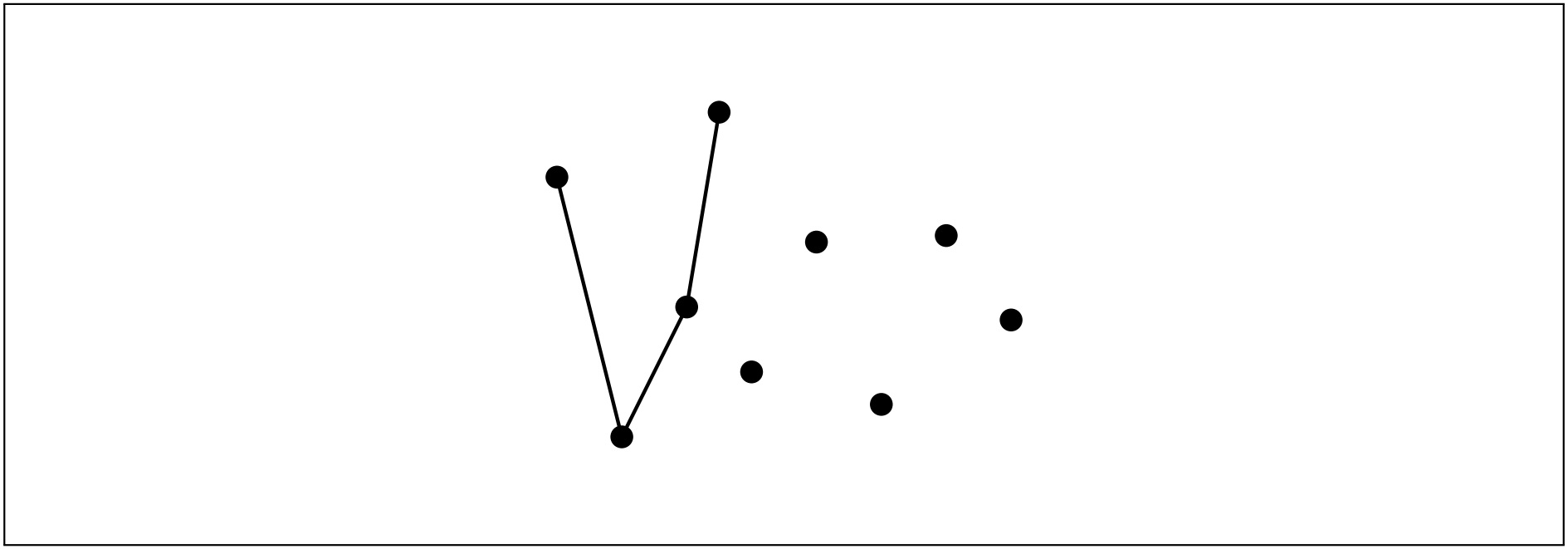
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



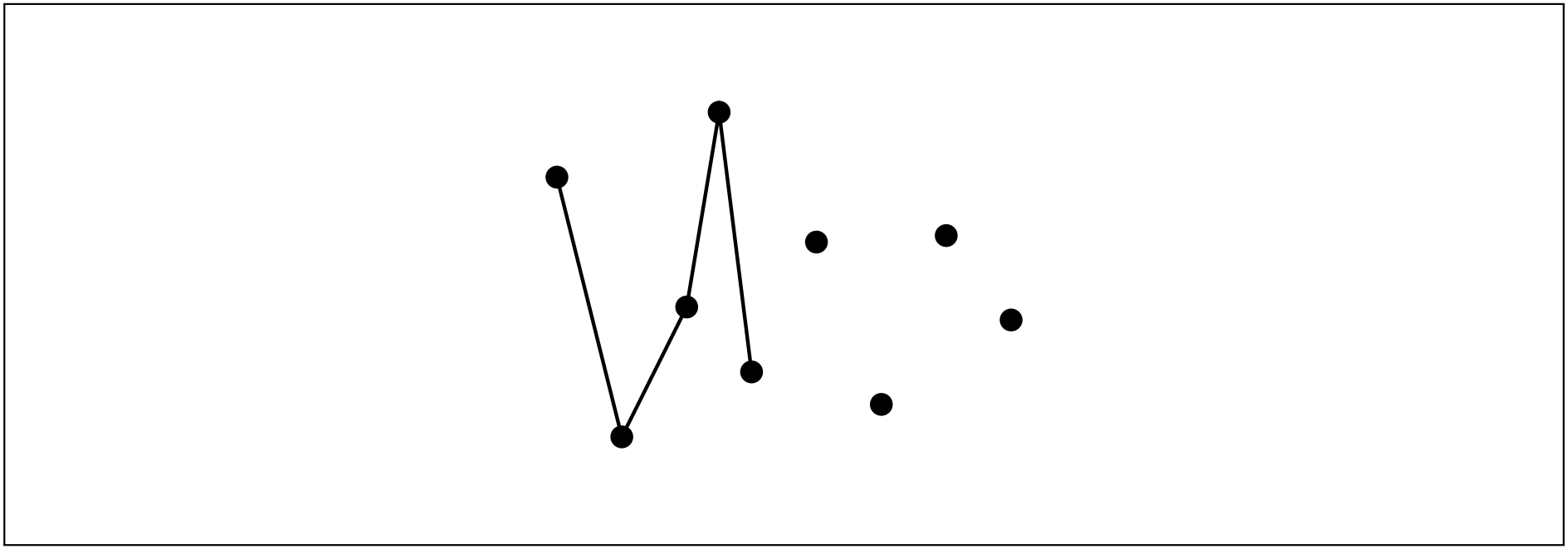
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



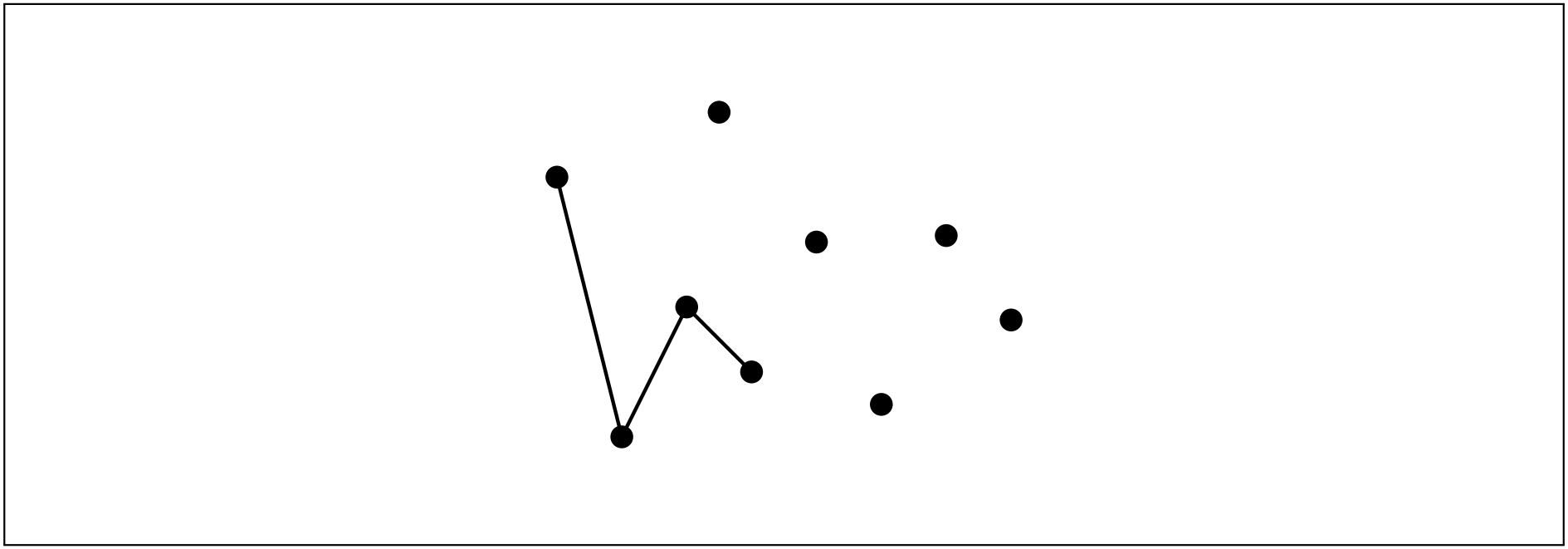
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



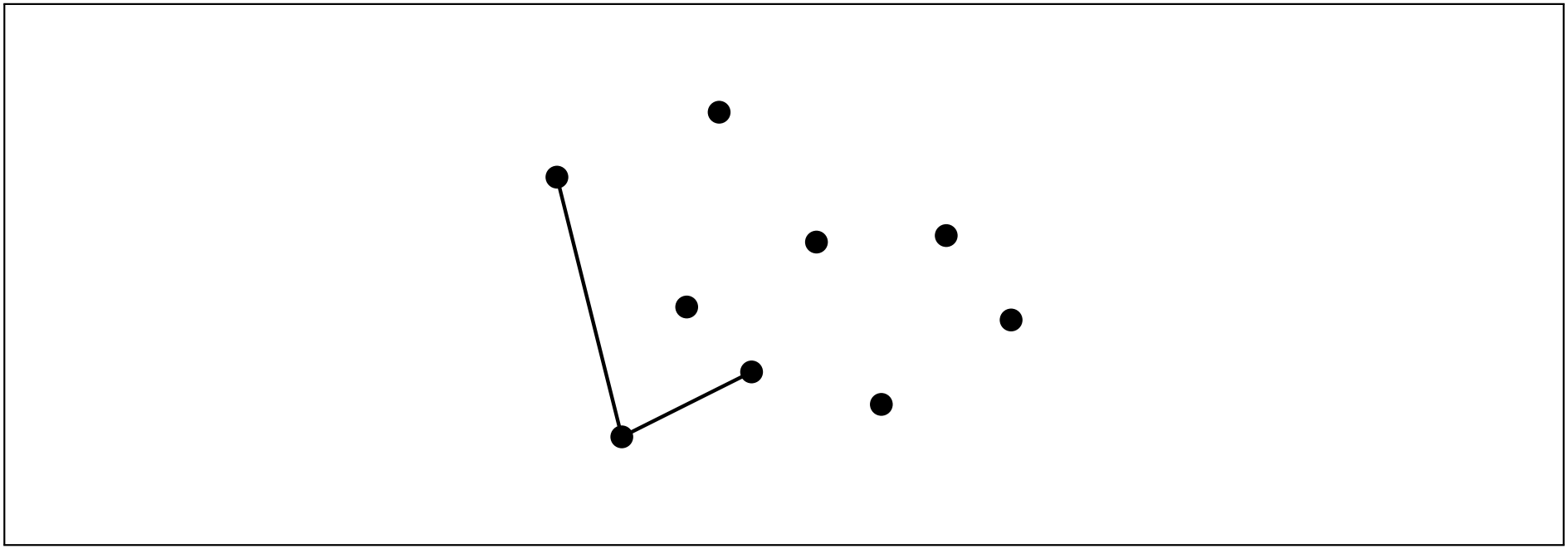
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



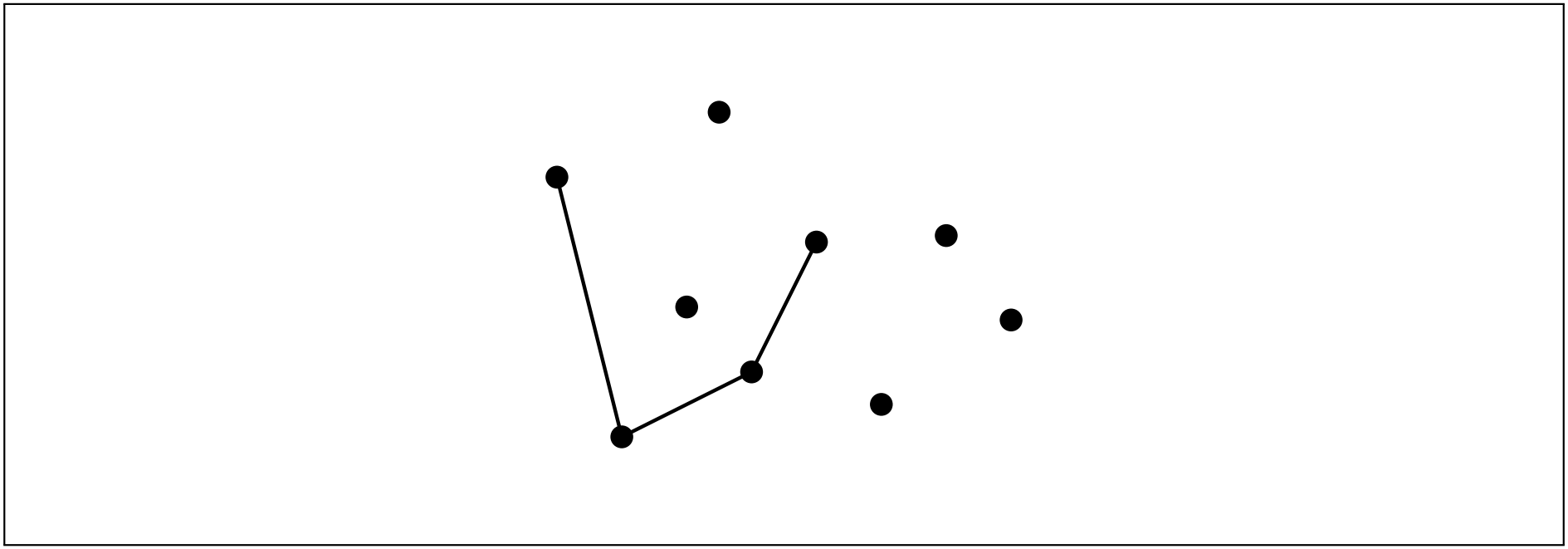
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



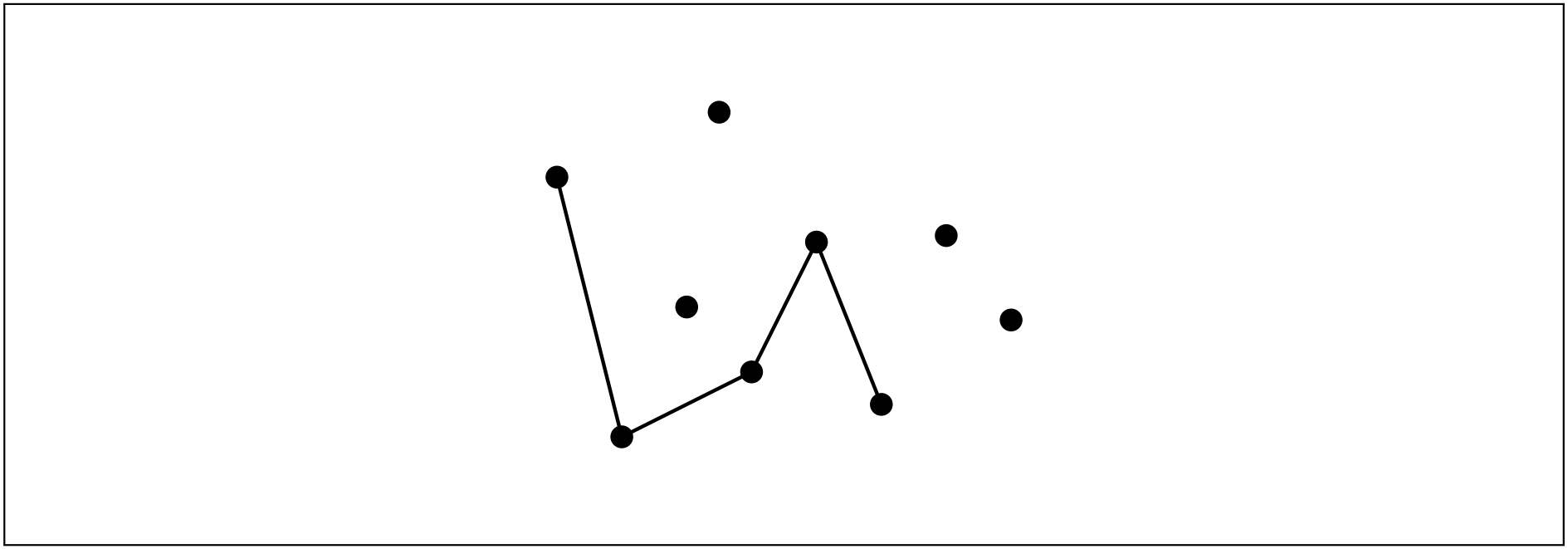
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



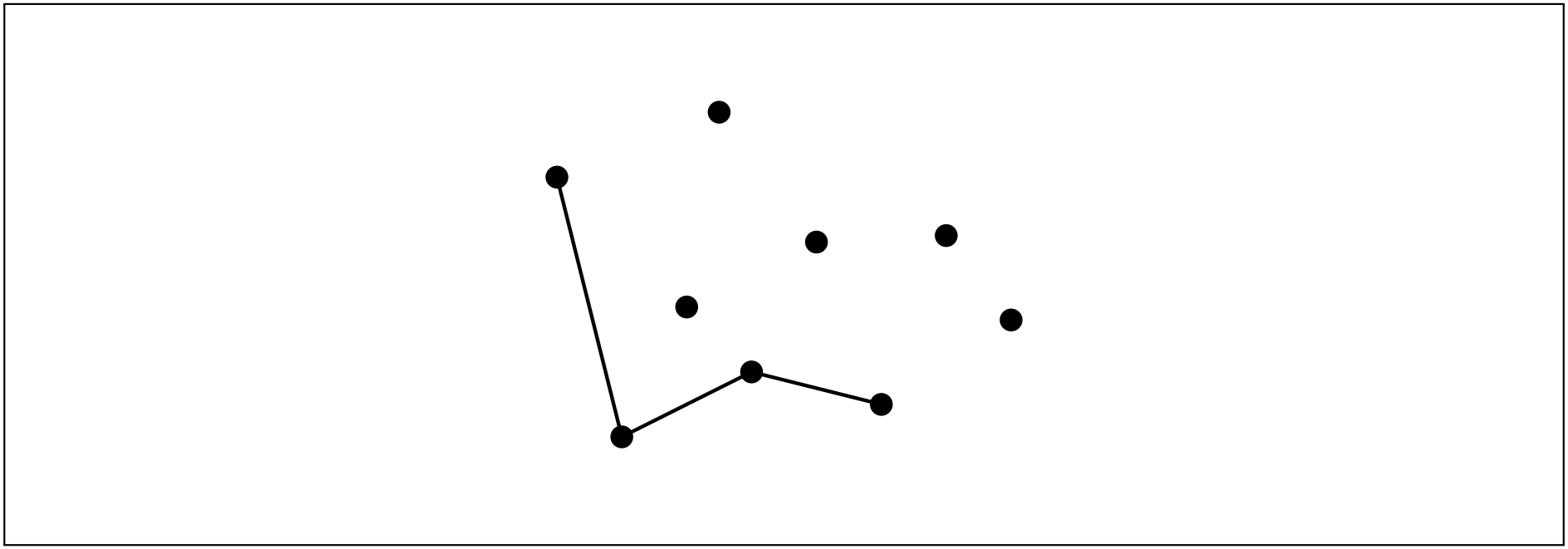
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



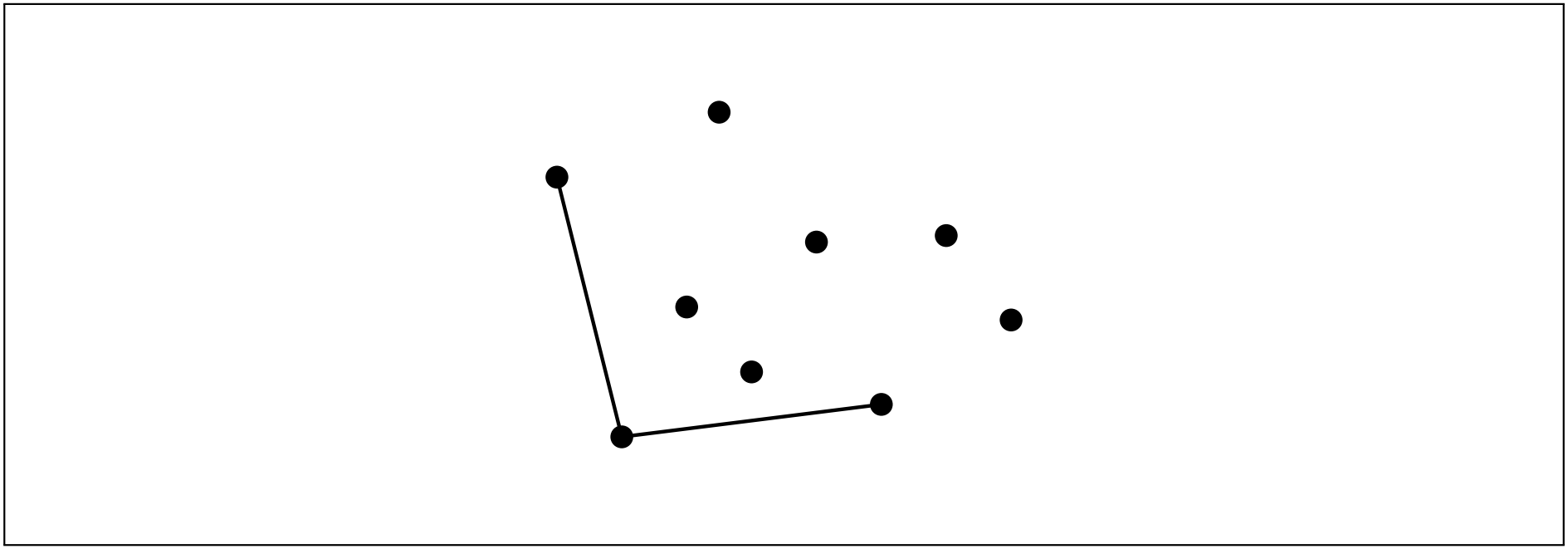
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



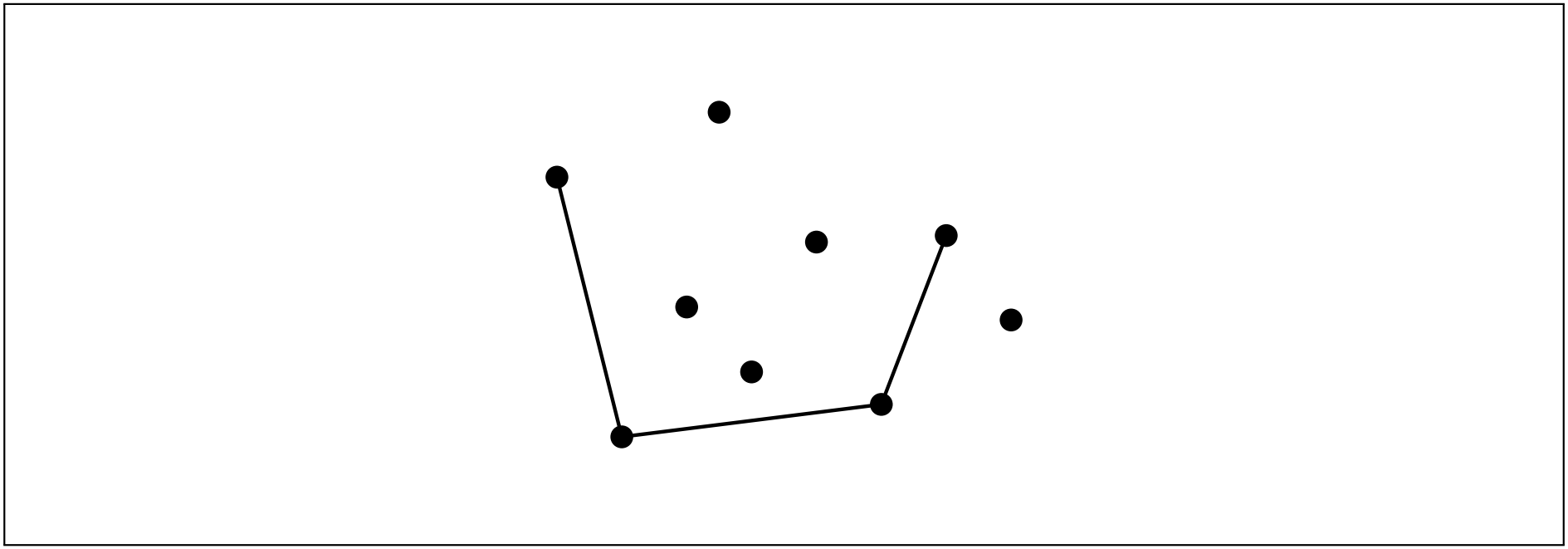
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



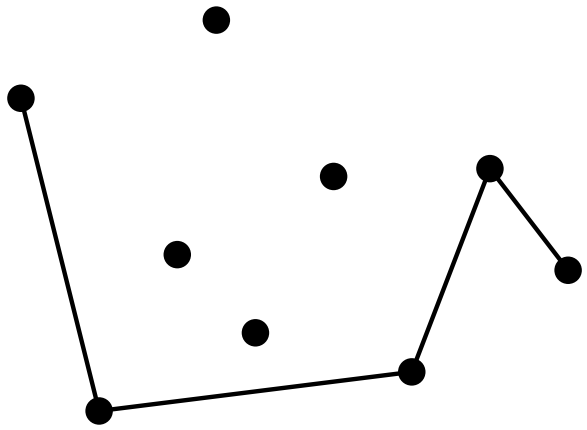
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



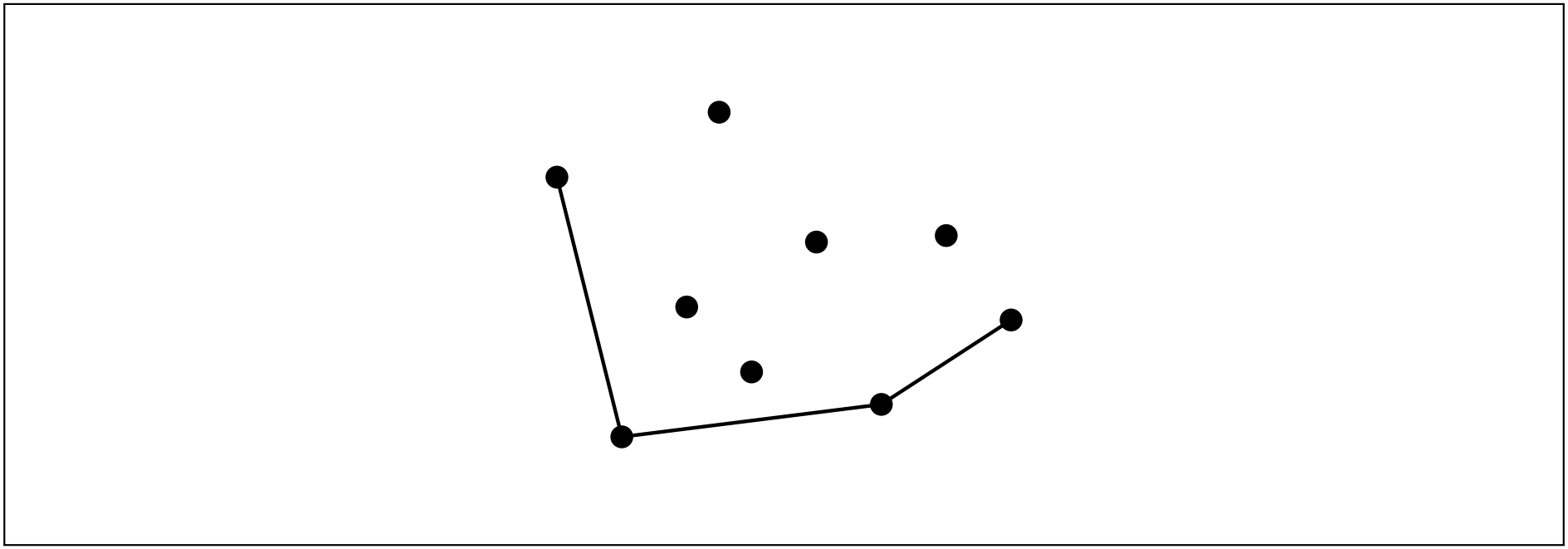
凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



凸包の計算方法

- 下側と上側に分けて考える
- 下側は x 座標の小さい頂点から走査
- 右回りでなくなったら頂点を削る



凸包を求める

```
ps.sort()
lower = []
for p in ps:
    while len(lower)>=2 and area(lower[-2],lower[-1],p)<0:
        lower.pop()
    lower.append(p)

upper = []
for p in reversed(ps):
    while len(upper)>=2 and area(upper[-2],upper[-1],p)<0:
        upper.pop()
    upper.append(p)

ch = lower[1:]+upper[1:]
l=[x for p in ch for x in p]
canv.create_polygon(*l,fill='white',outline='black')
```

- 「lower[1:]」 lower の最初の要素を除いたリスト
- 「*[x for p in ch for x in p]」 リストを引数の形に展開

応用：アニメーションと物理シミュレーション

ボールがはねる

```
import Tkinter

root = Tkinter.Tk()
canv = Tkinter.Canvas(root, width = 800, height = 600)
canv.pack()

def move(ball,v):
    canv.move(ball,0,v) # ball を (0,v) だけ移動
    (x1,y1,x2,y2) = canv.coords(ball) # ball の座標を取得
    v += 0.1
    if y2>400:
        v*=-0.9 # 反発係数 0.9
        canv.move(ball,0,400-y2)
    root.after(20,move,ball,v)

r=20
ball = canv.create_oval(400-r,100-r,400+r,100+r,fill='red')
canv.create_rectangle(0,400,800,600,fill='black')
root.after(100,move,ball,0) # mainloop 開始後 100ms 後に move(ball,0) を呼び出す
root.mainloop()
```

アウトライン

1. 文字列操作
2. 平面幾何
3. 演習

演習問題提出方法

演習問題は，解答プログラムをまとめたテキストファイル (***.txt) を作成して，OCW-i で提出して下さい.

- 提出期限は次回の授業開始時間です.
- どの演習問題のプログラムかわかるように記述してください.
- 出力結果もつけてください.
  - 描画する問題の場合はどのような結果が得られたか一言で説明
- おまけ問題をやる必要はないです.

演習問題

問 1

'20160620' のような年月日の 8 桁からなる文字列 (yyyymmdd) を引数として，'2016/6/20' のような文字列 (yyyy/mm/dd) を出力する関数を作成せよ．逆に，年月日がそれぞれ整数 (year,month,day) として与えられた時，8 桁の文字列に変換する関数も作成せよ．($0 \leq \text{year} \leq 9999$ と仮定する)

問 2

時刻にマッチするような正規表現を作成し，以下の入力で確かめよ

- マッチする: 「2:03」 「07:30」 「13:12」 「23:57」 「0:00」
- マッチしない: 「24:00」 「3:67」 「30:30」 「10:0」 「1a:00」

演習問題

問 3

ランダムに線分を何本かつくり，どれとも交差していない線分は黒で，どれかと交差している線分は赤で表示せよ．

問 4

凸包のプログラムを，凸包の面積の表示もするように改造せよ．
面積の表示には `create_text` を用いよ．

問 5 (おまけ)

凸包の計算を 1 ステップずつ描画するようなアニメーションを作成せよ．