- **Python + 基本**

C++萬用標頭檔:

Include<bits/stdc++.h>

Python:

- `string.find(substring)`: 尋找子字串在原字串中的位置，如果找不到則返回 -1。
- `string.index(substring)`: 尋找子字串在原字串中的位置，如果找不到會引發 ValueError。
- `string.startswith(prefix)`: 檢查字串是否以指定的前綴開始。
- `string.endswith(suffix)`: 檢查字串是否以指定的後綴結束。
- `string.count(substring)`: 返回子字串在原字串中出現的次數。

2. 字串判斷：
   - `string.isalpha()`: 檢查字串是否全部由字母組成。
   - `string.isdigit()`: 檢查字串是否全部由數字組成。
   - `string.isalnum()`: 檢查字串是否由字母和數字組成。
   - `string.islower()`: 檢查字串是否全部為小寫字母。
   - `string.isupper()`: 檢查字串是否全部為大寫字母。
   - `string.isspace()`: 檢查字串是否全部由空白字符組成。
3. 字串格式化：
   - 字串插值：使用 f-strings 或 `.format()` 方法來插入變數或值到字串中。
   - `string.capitalize()`: 將字串首字母轉換為大寫。
   - `string.title()`: 將字串中每個單詞的首字母轉換為大寫。
   - `string.center(width)`: 將字串置中並使用指定寬度的空格填充。
   - `string.ljust(width)`: 將字串左對齊並使用指定寬度的空格填充。
   - `string.rjust(width)`: 將字串右對齊並使用指定寬度的空格填充。

- `len(string)`: 回傳字串的長度。
- `string.lower()`: 將字串轉換為小寫。
- `string.upper()`: 將字串轉換為大寫。
- `string.strip()`: 移除字串兩側的空白字符。
- `string.split(separator)`: 以指定的分隔符將字串分割成列表。
- `string.join(iterable)`: 將可迭代物件中的元素用指定的字串連接起來。
- `string.replace(old, new)`: 將字串中的舊字串替換為新字串。

- **DFS:**

```
void DFS(bool* pass, vector<int>* vertex, int u) {
	pass[u] = true;
	for(auto v:vertex[u]){
		if(!pass[v]){
			DFS(pass, vertex, v);
		}
	}
}
```

- **Flood Fill:**

走過四周，如果可以走進繼續走。

- **finding bridges:**

首先有 low, dfn, time
low 紀錄可以走到最小值
dfn 記錄我哪時候走到他
然後每次 DFS 時 time 都要++
time++;
dfn[u] = low[u] = time;
然後再去看 u 的所有小孩 v
如果 v 還沒走過就去走
走完看 v 的 low 值跟 u 的 low 值哪個低 u 的 low 值就要換成那個 代表 u 也可以走到最祖先 low 那邊
然後 check v 的 low 值, 如果 u 的 dfn 值小於 v 的 low 值 代表 v 沒有一條路可以走回到比 dfn 更祖先的地方 代表 u-v 是一條 bridge
但如果 v 已經被走過的話就看看他的 dfn 值有沒有小於 u 的 low 值 有的話 u 的 low 值就要換 代表 u 可以走到更祖先(前提是這個是小孩不是父母)

- **finding artculation points:**

low 跟 dfn 跟 bridge 一樣
只要 v.low == v.dfn 或者 v.low == u.dfn 代表這個 v 點最高只能到自己或是自己的祖先 那這個點就是 artculation points

- **bridge connected:**

一張無向圖上，不會產生橋的連通分量，稱作「橋連通分量」。

- ## MST(最小生成數):

  krkruskal 就是找老大。

- ## Dp(動態規劃問題):

  https://blog.csdn.net/tengfei461807914/article/details/47031373

- ## convex hull(凸包):

  若點尚未排序，先排序點的順序

```
bool cmp1(Vertex a, Vertex b) {
        if(a.y == b.y) return a.x < b.x;
        else return a.y < b.y;
}
bool cmp2(Vertex a, Vertex b) {
        int cross = cross_product(vertex[0], a, b);
        return cross > 0 || (cross == 0 && dist(vertex[0], a) < dist(vertex[0], b));
}
```

  cmp1 找出最左下的點，cmp2 找出與 vertex[0]的角度排序

  排序完後 vertex 要再 push_back 一個 vertex[0]

  創一個空的 s 來放最後的答案

```
for(int i = 0 ; i < vertex.size() ; i++) {
          int m = s.size();
          while(m >= 2 && cross_product(s[m-2], s[m-1], vertex[i]) <= 0) {
                s.pop_back();
                m--;
          }
          s.push_back(vertex[i]);
   }
```

  若只是要判斷是否為凸包就只需要看每連續三個點的 cross 角度是否同個方向即可

- ## catalan number(卡特蘭數):

  是那種遞迴方程式長這樣 $c_{n+1} = c_0c_n + c_1c_{n-1} + \cdots + c_nc_0, \ \ c_0 = 1$

  解完遞迴之後有一般式為 $c_n = \frac{1}{n+1}\binom{2n}{n}$

  寫程式看是要用遞迴方程式或是一般式都可以

- **maxflow(最大流):**

先建立一個表 G[n][n]裡面放流量，然後我覺得你看程式碼就會懂...

```
void maxflow(int n) {
    while(true) {
        queue <int> q;
        q.push(s);
        int flow[n*2+2] = {0};
        flow[s] = INT_MAX;
        int p[n*2+2]; //parents
        for(int i = 0 ; i <= n*2+1 ; i++) p[i] = -1;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int v = 1 ; v <= n*2+1 ; v++) {
                if(!flow[v] && G[u][v] > 0) {
                    p[v] = u;
                    flow[v] = min(flow[u], G[u][v]);
                    q.push(v);
                }
            }
        }
        if(!flow[t]) break;
        for(int v = t ; v != s ;) {
            int u = p[v];
            G[u][v] -= flow[t];
            G[v][u] += flow[t];
            v = u;
        }
        f += flow[t];
    }
}
```

然後還有可能會拆點

- # 不規則多邊形:

逆時鐘排序(適用於凸包):

1. 先找出重心 center

```
For(int I = 0 ; I < n ; i++) {
        sumX += p.x;
        sumY += p.y;
}
Return (sumX/n + sumY/n);
```

2. 極角計算

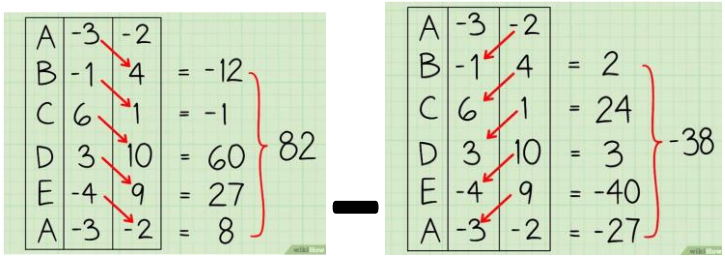
```
double polarAngle(const Point& p1, const Point& p2) {
        return atan2(p2.y - p1.y, p2.x - p1.x);
}
```

3. 排序

```
bool cmp(Point a, Point b) {
        double angel1 = polarAngle(center, a);
        double angel2 = polarAngle(center, b);
        return angel1 < angel2;
}
```

面積計算方式:

把點按照順時針或是逆時針排序之後，多加一個 point[0]在最後

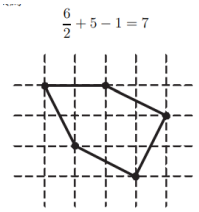



```
for(int i = 0 ; i < n ; i++) {
      area += point[i].x * point[i+1].y;
      area -= point[i].y * point[i+1].x;
}
area/=2;
```

Pick 公式:

假設平面上有一個以格子點爲頂點之凸多邊形, 則其面積爲

$$\frac{a}{2}+b-1$$



a 是邊界經過的邊，b 是內部點

a 的算法:先求每條邊的絕對值 dx, dy(a.x-b.x, a.y-b.y)，求 gcd(dx, dy)

求出的 gcd 值-1 即是 a 值

- # 數學思維:

1. 如果一個向量空間中的 n 個向量的夾角都小於等於 π ，那麼它們的線性組合可以為零向量。
2. 尤拉公式: 在 planar 中，V-e+r=2，其中 V 代表點數 e 代表邊數 r 代表 region 數。
3. 費氏數列:常常看到費氏進制，2=10(fib)、3=100(fib)等等。

- # IOU:

登录后复制

```
struct bbox
{
    int m_left;
    int m_top;
    int m_width;
    int m_height;

    bbox() {}
    bbox(int left, int top, int width, int height)
    {
        m_left = left;
        m_top = top;
        m_width = width;
        m_height = height;
    }
};

float IOU_compute(const bbox b1, const bbox b2)
{
    w = max(min((b1.m_left + b1.m_width), (b2.m_left + b2.m_width)) - max(b1.m_left, b2
    h = max(min((b1.m_top + b1.m_height), (b2.m_top + b2.m_height)) - max(b1.m_top, b2.

    return w*h / (b1.m_width*b1.m_height + b2.m_width*b2.m_height - w*h);
}
```

```
登录后复制

t;
;
th;
ght;


left, int top, int width, int height)

t = left;
 = top;
th = width;
ght = height;



pute(const bbox b1, const bbox b2)

in((b1.m_left + b1.m_width), (b2.m_left + b2.m_width)) - max(b1.m_left, b2.m_left), 0);
in((b1.m_top + b1.m_height), (b2.m_top + b2.m_height)) - max(b1.m_top, b2.m_top), 0);

h / (b1.m_width*b1.m_height + b2.m_width*b2.m_height - w*h);
```

- **尤拉迴路&尤拉路徑:**

無向圖:

把所有的 vertex 的邊數算出來，前提是要連通，假如全都是偶數的話那就有尤拉迴路，如果有兩個奇數的話就有尤拉路徑。

無向圖:

把所有的 vertex 的 indegree 跟 outdegree 算出來，前提是要連通，假如全都是 0 的話那就有尤拉迴路，如果有一個 1 一個-1 且其他都是 0 的話就有尤拉路徑。

Hints:

```
pair<int, int> edge;
set<int> s;
void DFS(int u) {
s.insert(u);
for(auto v:G[u]) {
    for(int j = 0 ; j < edge.size() ; j++) {
```

```
                if( (edge[j] == make_pair(u, v) || edge[j] == make_pair(v,u))
                    && visited[j] == false) {
                    visited[j] = true;
                    ans.push_back(v);
                    DFS(v);
                }
            }
        }
}
```

判斷圖有無連通:

跑 DFS 或 BFS 都可以，在跑的時候加上一個 set 來記錄跑過的點，最後在看這個 set 的大小是否與 vertex 數一樣即可。