

0.1 Détection d'ensembles d'une même couleur

Une autre approche pour détecter des pièces de monnaie sur une image est d'exploiter le contraste entre les pièces elles-mêmes et le matériau sur lequel elles sont placées. Dans l'image suivante, par exemple, les pièces sont jaune ou rouge foncé alors que le fond est blanc.

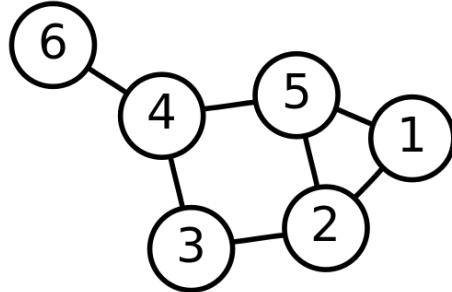


Nous allons développer un algorithme permettant d'utiliser la couleur du fond pour isoler les pièces. Et nous allons l'appliquer à cette image afin d'en extraire les pièces.

1 Un algorithme en profondeur

1.1 Généralités sur les graphes

Un graphe est un ensemble de points appelés *sommets* (*vertices* en anglais) et de connections appelées *arêtes* (*edges* en anglais). Voici par exemple un graphe de 6 sommets et de 7 arêtes :



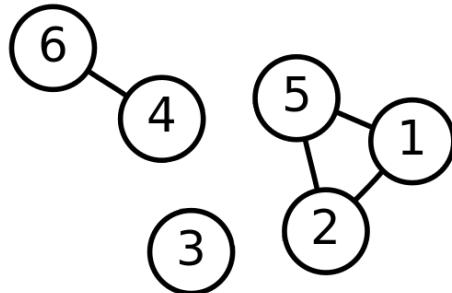
Dans ce cas-ci, les points sont 1, 2, 3, 4, 5, 6 et les connections sont (1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6).

D'un point de vue mathématique, le graphe G peut se définir ainsi :

$$\begin{aligned} G &= (V, E), \\ V &= \{1, 2, 3, 4, 5, 6\}, \\ E &= \{(1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6)\}. \end{aligned}$$

1.2 Composantes connexes

Le graphe ci-dessus est un exemple de graphe *connexe*, c'est-à-dire un graph dont tout point est relié à tout point. Il existe aussi des graphes non connexes. Les différentes parties connexes du graphe sont alors appelées des *composantes connexes*. Le graphe ci-dessous est ainsi composé de 3 composantes connexes. Il est clair que le graphe est non connexe car le point 1 n'est par exemple pas relié au point 3.



1.3 Parcours d'un graphe

Voyons maintenant comment on peut détecter les différentes composantes connexes d'un graphe. Pour ce faire, commençons par remarquer la transitivité de la définition d'une composante connexe. En effet, s'il existe un chemin de u à v et de u à w , il existe forcément un chemin allant de v à w et inversement. Il suffit en effet de partir de v , de se rendre à u et de rejoindre w depuis u . Cette propriété peut sembler anodine, mais elle affirme que pour détecter une

composante connexe, il suffit de partir d'un sommet quelconque et de parcourir le graphe en cataloguant tous les sommets atteignables.

Un algorithme classique pour parcourir les graphes est le *depth-first search* (DFS) ou encore *parcours en profondeur*, appelé ainsi parce qu'il va aussi loin qu'il peut avant de revenir en arrière.

On part d'un sommet u . Si le sommet u n'a pas encore été visité, on le marque comme visité. Ensuite, pour chaque sommet v adjacent à u , on reprend l'algorithme récursivement. En C++, cela donne :

```

1 const int N; // nombre de sommets (numérotés de 0 à N - 1)
2
3 // Pour chaque sommet, on garde une liste des sommets adjacents
4 vector<int> adjacent_vertices[N];
5
6 // Au départ, aucun des sommets n'a été visité
7 bool visited[N] = {false};
8
9 void dfs(int u) {
10    if (!visited[u]) { // Si u n'a pas été visité
11        visited[u] = true;
12        // Pour chaque sommet v adjacent
13        for (int v : adjacent_vertices[u]) {
14            dfs(v); // On continue récursivement
15        }
16
17        // Cataloguer u d'une façon ou d'une autre
18        /* ... */
19    }
20}

```

Listing 1 – DFS

Cet algorithme ne parcourt qu'une seule composante connexe. Afin de détecter toutes les composantes connexes, on peut essayer de lancer l'algorithme depuis chaque sommet. Si l'algorithme n'a pas encore été lancé depuis un sommet u , c'est que u appartient à une nouvelle composante connexe.

```

1 // Pour chaque sommet u
2 for (int u = 0; u < N; ++u) {
3    if (!visited[u]) {
4        // Nouvelle composante connexe
5        // donc on appelle l'algorithme.
6        dfs(u);
7    }
8}

```

Listing 2 – Composantes connexes

Ce code parcourt bien tous les sommets mais il ne permet toujours pas de déterminer la composante connexe d'un sommet donné. Réglons ce problème : plutôt que d'enregistrer si un sommet a été visité ou non, on peut directement enregistrer à quelle composante connexe il appartient.

```

1 // Au départ, aucun des sommets n'a été visité (cc[u] = -1 ∀u)
2 int cc[N];
3
4 void dfs(int u, int id) {
5     if(cc[u] == -1) { // Si u n'a pas été assigné
6         cc[u] = id;
7         // Pour chaque sommet v adjacent
8         for(int v : adjacent_vertices[u]) {
9             dfs(v, id); // On continue récursivement
10        }
11    }
12}
13
14 // On initialise à -1
15 for(int u = 0; u < N; ++u) cc[u] = -1;
16
17 // Composante connexe actuelle
18 int id = 0;
19 // Pour chaque sommet u
20 for(int u = 0; u < N; ++u) {
21     if(cc[u] == -1) {
22         // Nouvelle composante connexe
23         // donc on appelle l'algorithme.
24         dfs(u, id++);
25     }
26}

```

Listing 3 – Composantes connexes