Quelques problèmes ouverts

Alexandre Blondin Massé

Laboratoire de combinatoire et d'informatique mathématique
Université du Québec à Montréal

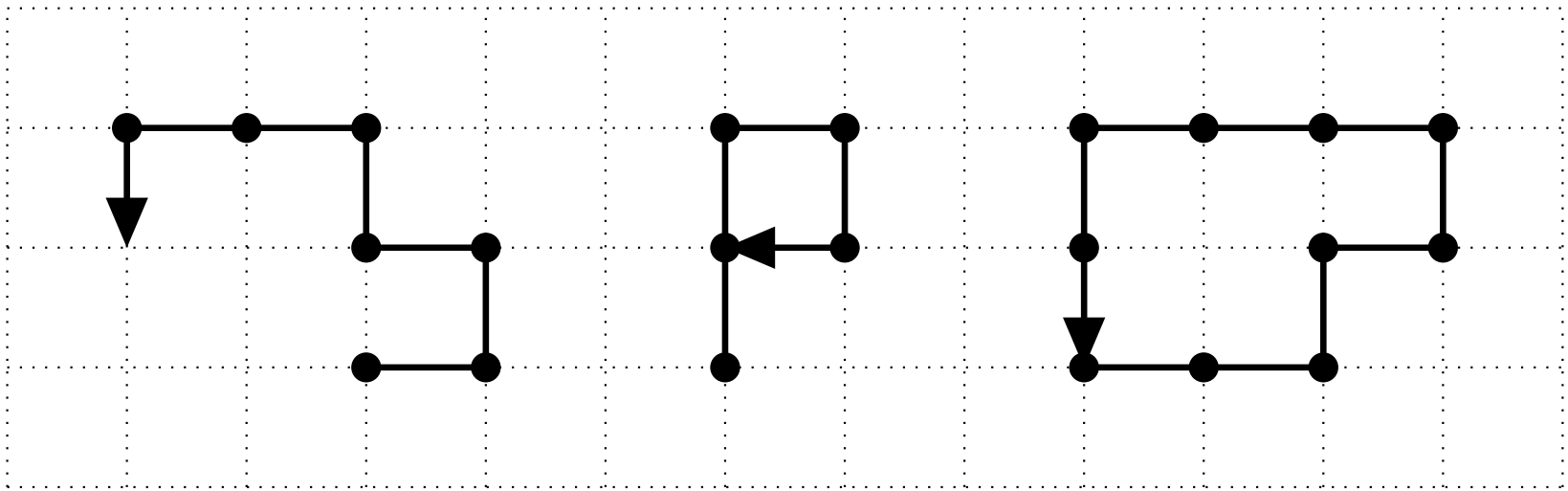
17 février 2017
INF7341 Structures de données

Plan

Plan

Plan

Chemins discrets

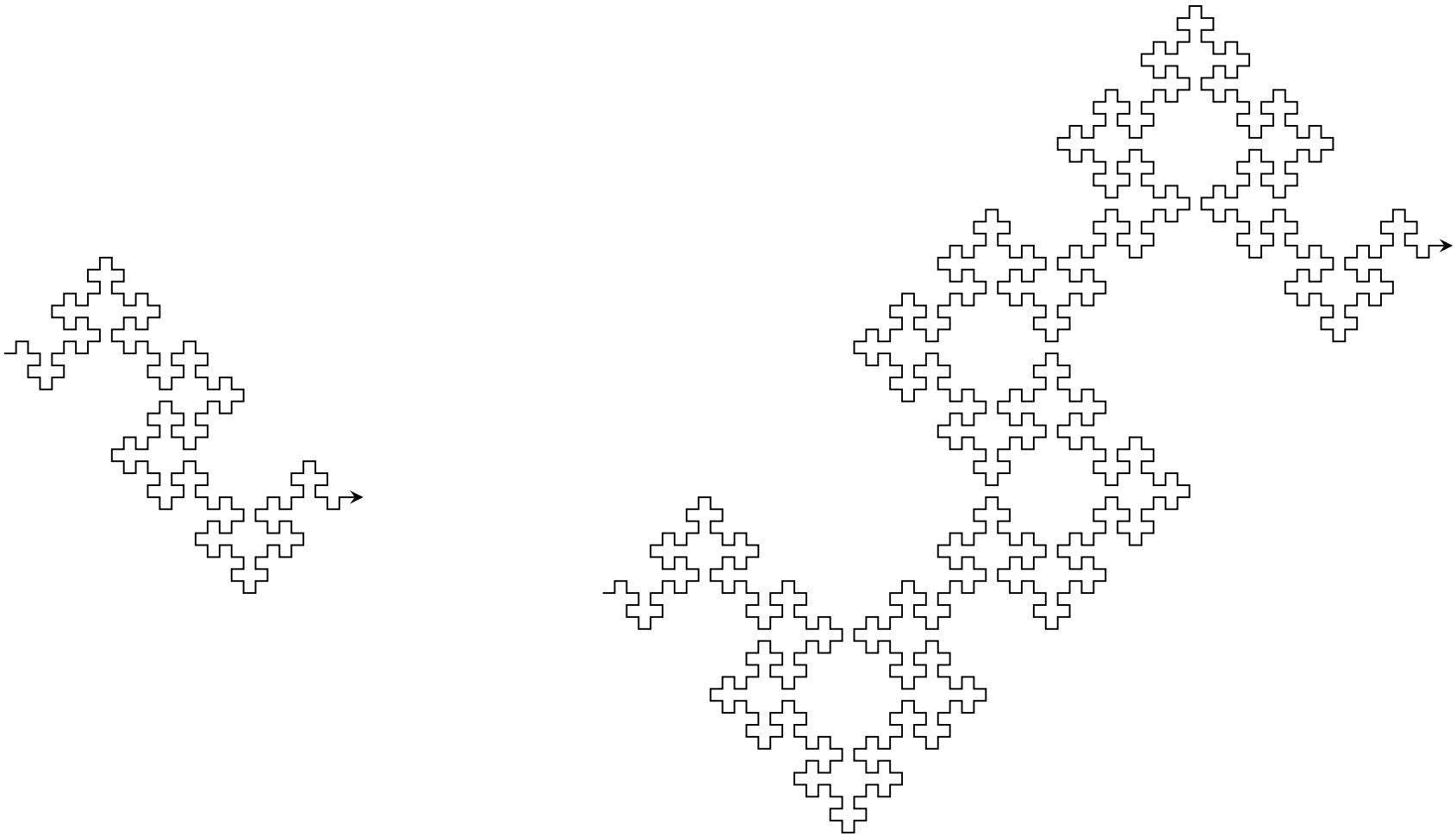


(a) (b) (c)

Trois **chemins discrets** :

- (a) **simple**;
- (b) **non simple**;
- (c) **fermé** et **simple**.

Chemin auto-évitant



Plan

Questions

Étant donné un **chemin discret**...

- ...est-il **simple**?
- ...combien a-t-il de **points d'intersection**?

Étant donné **deux régions discrètes** (chemin fermé)...

- ...quelle est leur **intersection**?
- ...quelle est leur **réunion**?
- ...quelle est leur **différence**?

Plus généralement, avec quelle **efficacité** peut-on répondre à ces questions?

Solution 1: arbre équilibré/en triant

- On parcourt le chemin, **étape par étape**;
- On insère les points visités dans un **arbre équilibré**;
- S'il y a un point d'intersection, il est aussitôt détecté puisque le point **se trouve déjà** dans l'arbre;
- **Complexité?**
  - $\mathcal{O}(n \log n)$ en temps;
  - $\mathcal{O}(n)$ en espace;
- De façon similaire, il est possible de stocker **tous les points** dans une liste et de les **trier**, pour obtenir les **mêmes complexités** dans le pire cas.

Solution 2: matrice booléenne

- En utilisant une **matrice booléenne**, on peut aussi **marquer** les points visités;
- Soit w et h la **largeur** et la **hauteur** du rectangle circonscrit contenant le chemin;
- Alors on a les **complexités** suivantes :
  - $\mathcal{O}(n)$ en temps;
  - $\mathcal{O}(wh)$ en espace.
- Il est facile de construire des exemples où $w, h \in \Omega(n)$, de sorte que la **complexité spatiale** est alors $\Theta(n^2)$.

Solution 3: table de hachage

- On **marque** les points visités à l'aide d'une **table de hachage**;
- Il n'y a malheureusement pas de garantie que chaque opération soit $\mathcal{O}(1)$ dans le **pire cas**.
- **Complexité?**
  - Temporelle : dépend de la **qualité** de la fonction de hachage;
  - Spatiale : dépend de la **capacité** de la table de hachage.
- Comme la **distribution** des chemins discrets n'est pas connue, il semble difficile d'imaginer une fonction de hachage qui se rapproche du **hachage parfait**.

Solution 4: tri linéaire

- Il est possible de **trier** une liste en temps **linéaire** :
  - *Bucket sort*;
  - *Counting sort*;
  - *Radix sort*.
- On trie d'abord les points par rapport à leur **abscisse** et ensuite par rapport à leur **ordonnée** (**tri radix**);
- **Complexité?**
  - Time : $\Theta(n)$;
  - Space : $\Theta(n)$.
- Cependant, l'**intersection** ne peut être détectée **en temps réel**.

La structure de données BKP

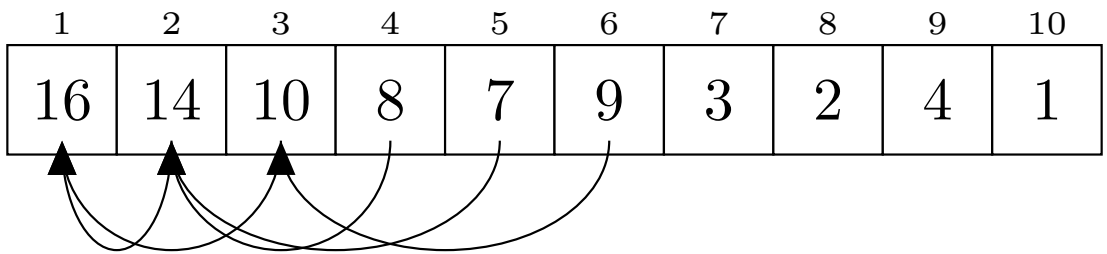
- S. Brlek, M. Koskas, X. Provençal, *A linear time and space algorithm for detecting path intersection in* $\mathbb{Z}^d$, Theoretical Computer Science (TCS) **412**, 2011, p. 4841-4850;
- Les auteurs introduisent une structure de données appelée **arbre radix enrichi**, qui permet de détecter les points d'intersection efficacement.
- Nous présentons cette structure d'abord sur $\mathbb{N}^2$, puis nous montrons comment elle peut être étendue à $\mathbb{Z}^2$.

La relation radix

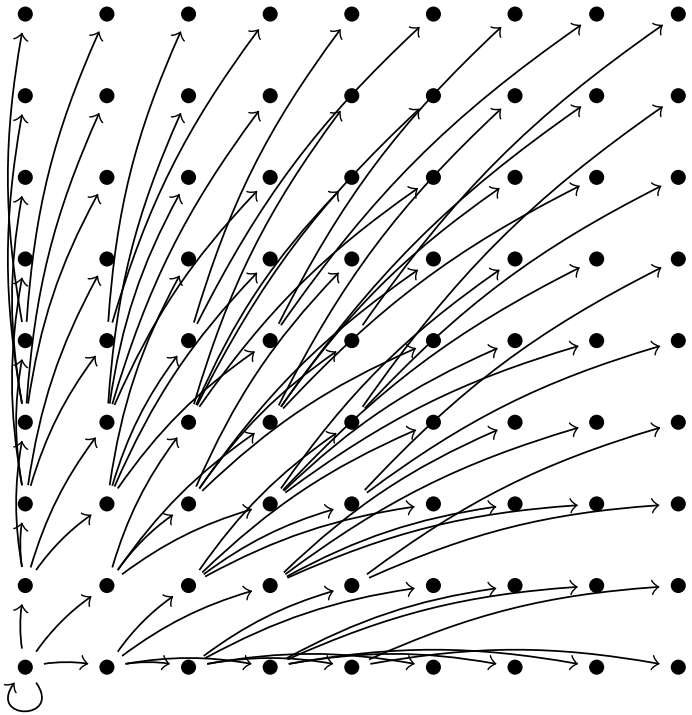
- La **relation radix** sur $\mathbb{N}^2$ est définie par

  $(x, y) \to (x', y')$ si et seulement si $(x, y) = \left(\lfloor x'/2 \rfloor, \lfloor y'/2 \rfloor\right)$

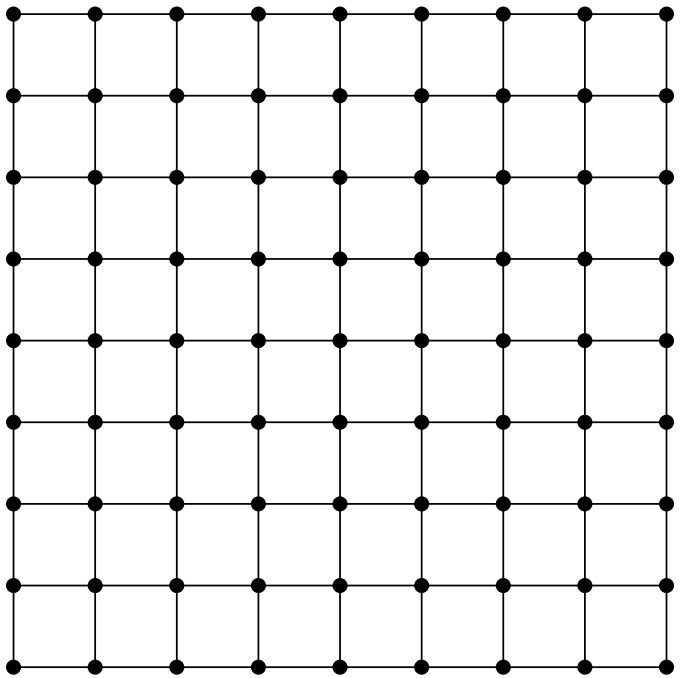
- Clairement, $\to$ est **antisymétrique**;
- (x, y) est appelé **parent** de (x', y');
- (x', y') est appelé **enfant** de (x, y);
- Il s'agit d'une **généralisation 2D** de l'ordre **monceau** pour les tableaux:



Combinaison de deux relations sur $\mathbb{N}^2$



(a)



(b)

- (a) La relation **radix**;
- (b) La relation de 4-**adjacence**;
- Nous devons comprendre l'**interaction** entre ces deux relations.

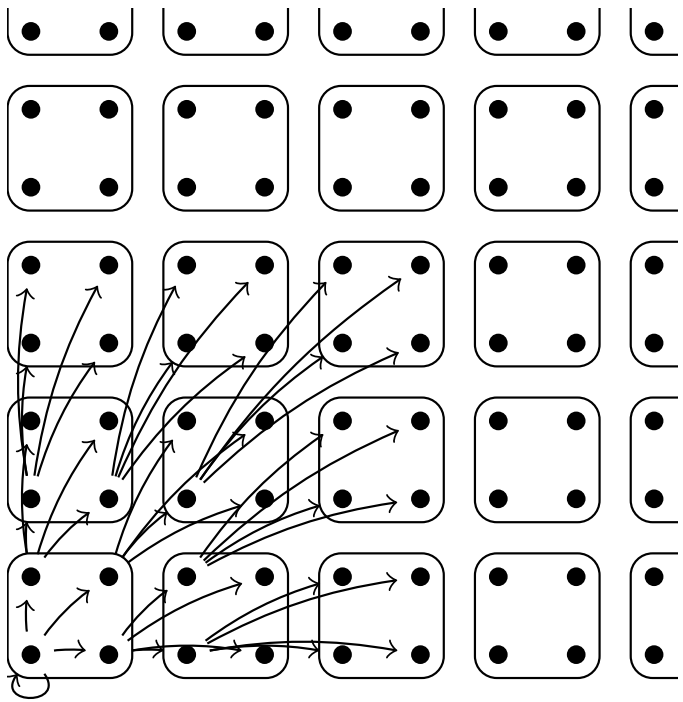
Observation 1

Lemme (Brek, Koskas, Provençal, 2011)

Si $p, p' \in \mathbb{N}^2$ sont **voisins**, alors exactement une des deux conditions suivantes est vérifiée :

(i) p et p' sont **frères**, c'est-à-dire parent (p) = parent (p');

(ii) parent (p) et parent (p') sont **voisins**, c'est-à-dire 4-adjacents.



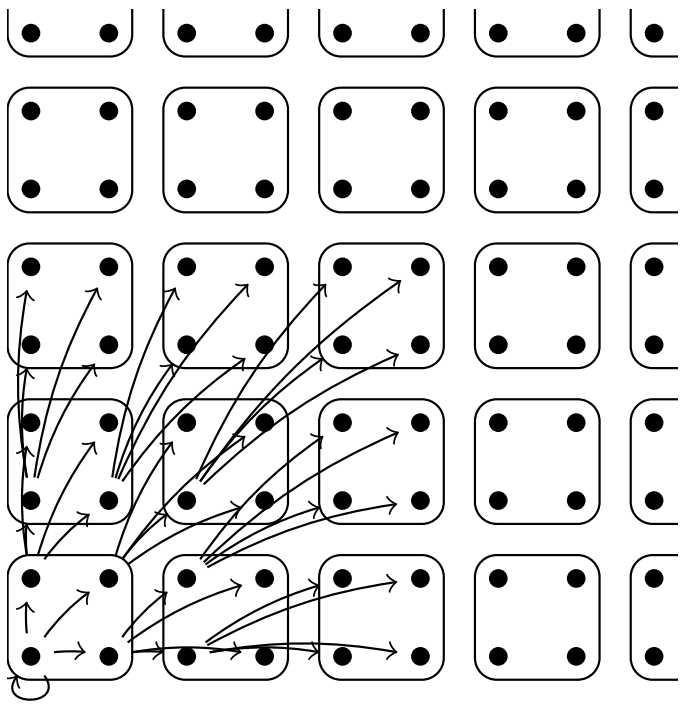
Observation 2

Lemme (Brlek, Koskas, Provençal, 2011)

Soient $p, p' \in \mathbb{Z}^2$ deux **voisins non frères**. Alors

$$\text{parent}\,(p') = \text{parent}\,(p) + \vec{d}\,,$$

où $\vec{d} = \overrightarrow{pp'} \in \{(1,0),(0,1),(-1,0),(0,-1)\}$.



Idée

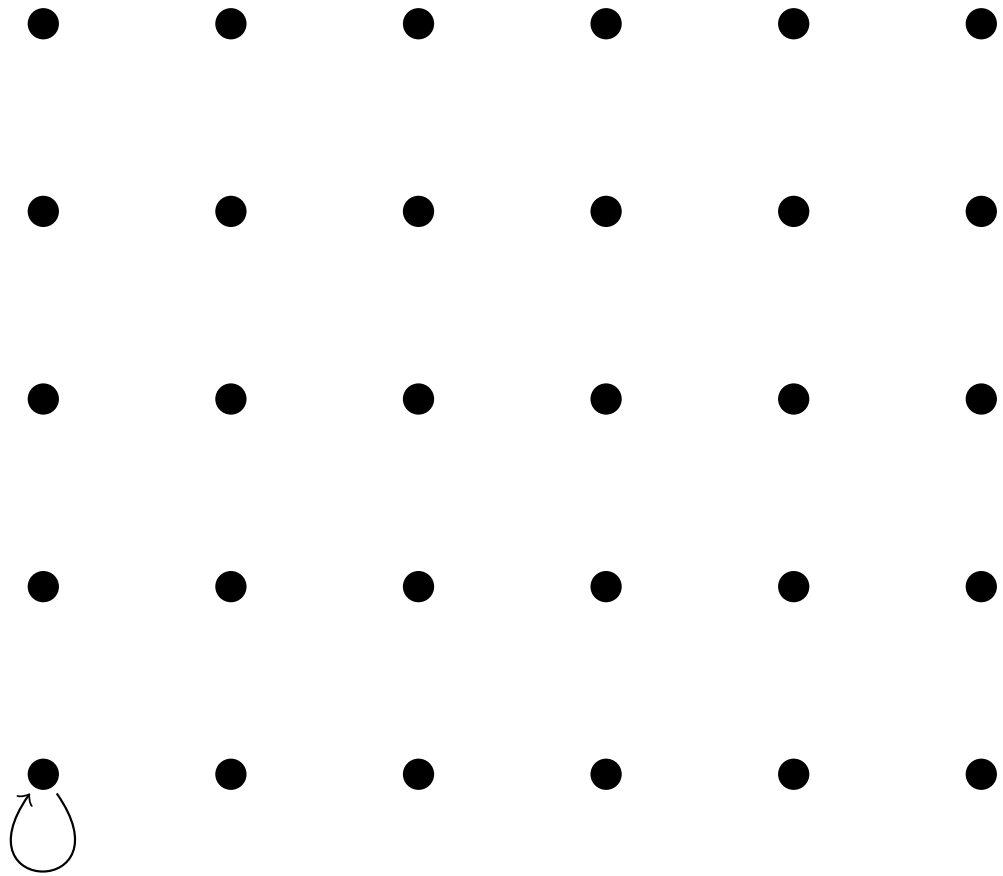
Lorsqu'on parcourt le chemin, on **ajoute**

- un **sommet** pour chaque **nouveau** point visité;
- des **arêtes** entre **certains voisins**;
- des **arcs** entre **certains points** liés par la relation **parent/enfant**;

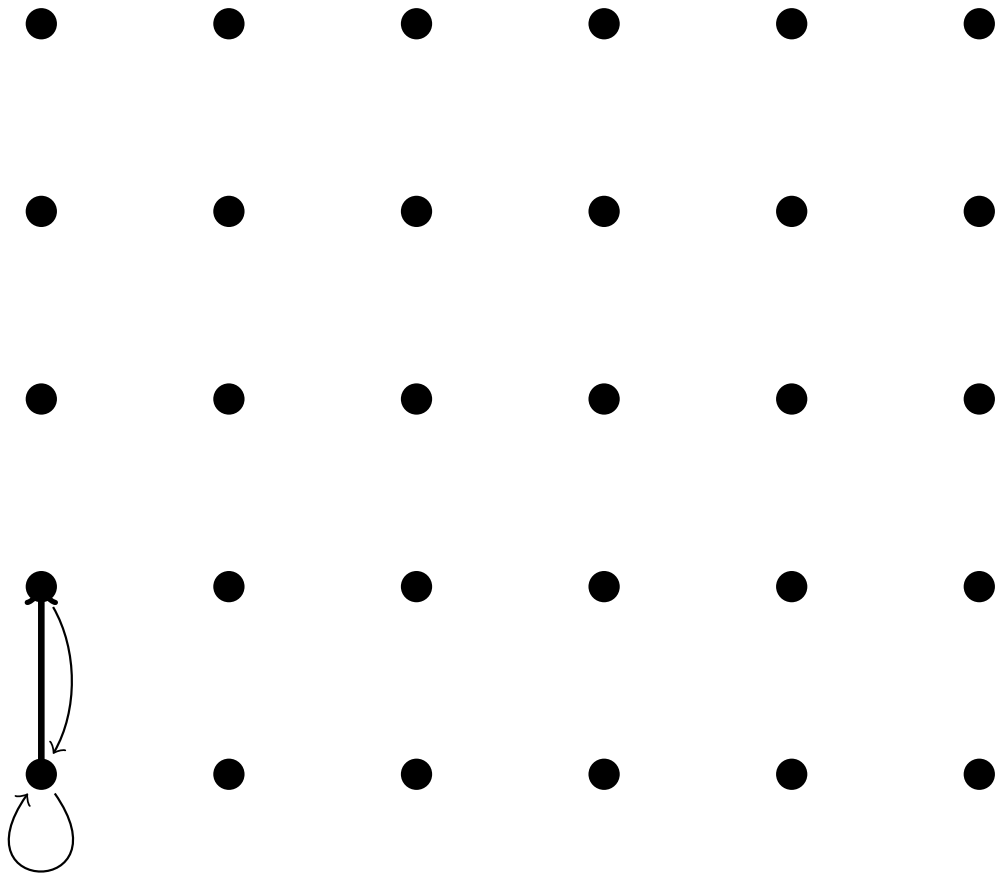
Puis

1. On démarre à l'**origine**;
2. On **lit** le prochain déplacement;
3. On **trouve** le voisin dans la structure de données, **en ajoutant** tout **sommet** ou **arc/arête** nécessaires;
4. On **répète** les étapes 2-3 jusqu'à la fin du chemin discret.

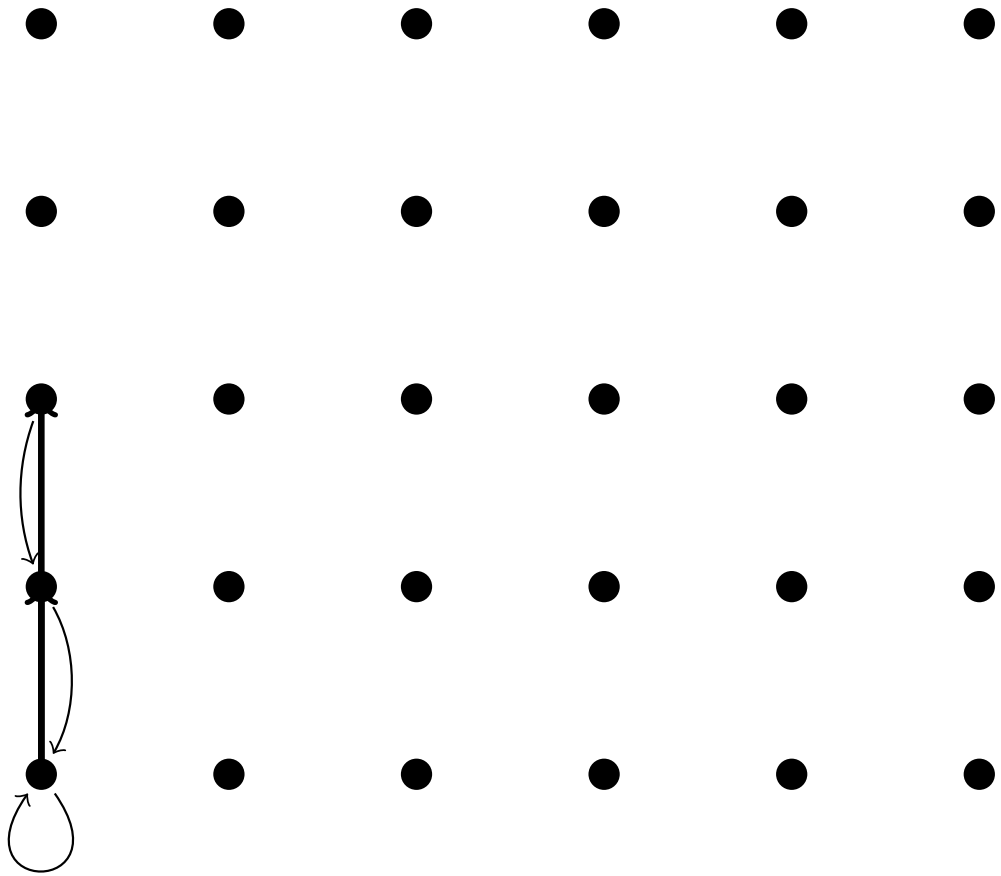
Exemple



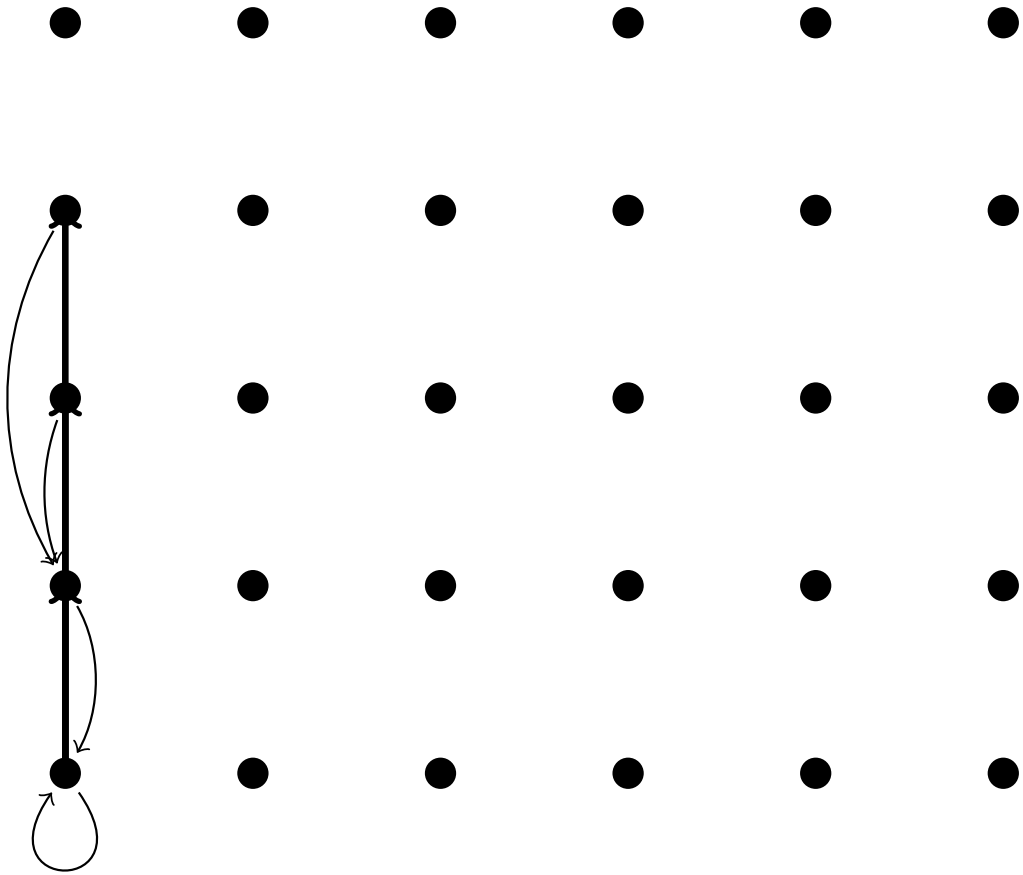
Exemple



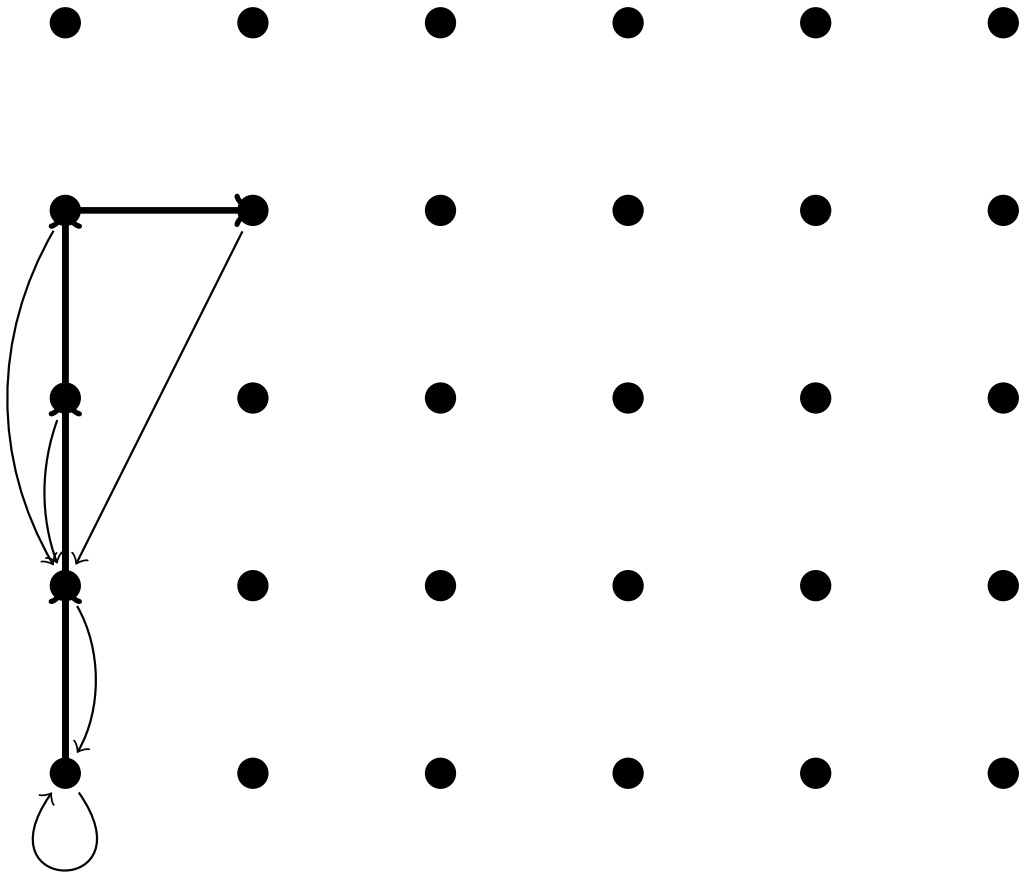
Exemple



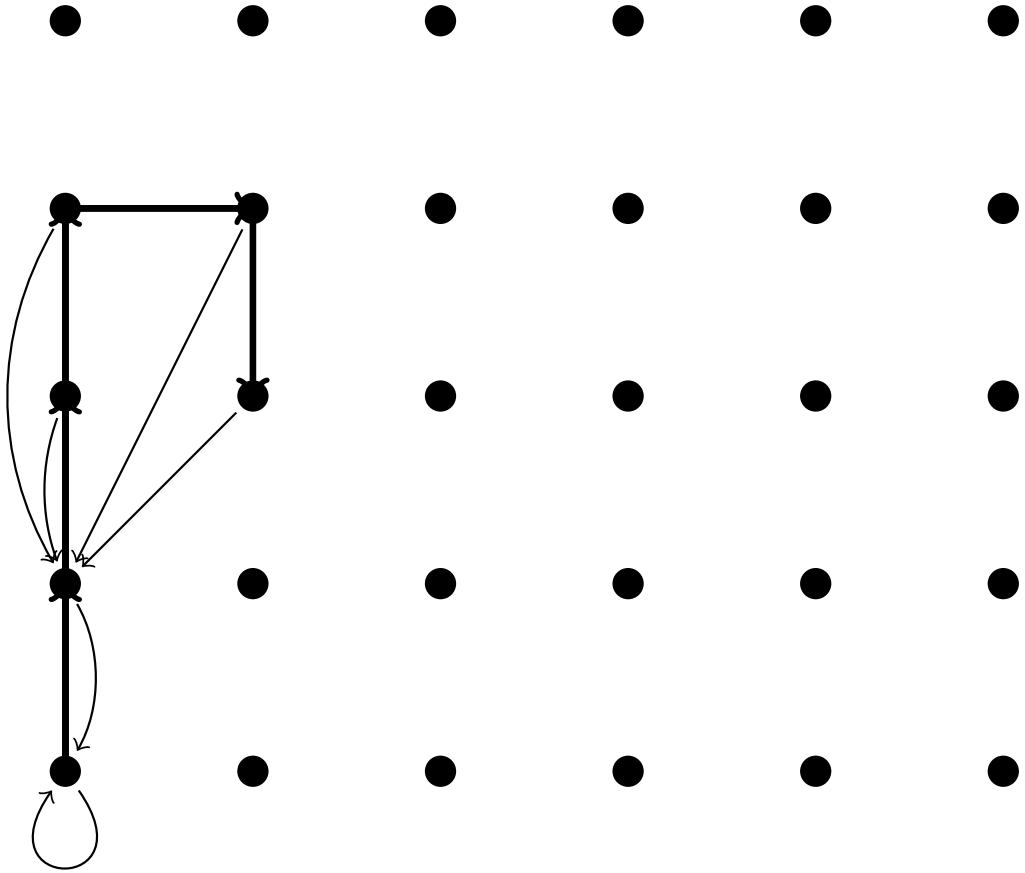
Exemple



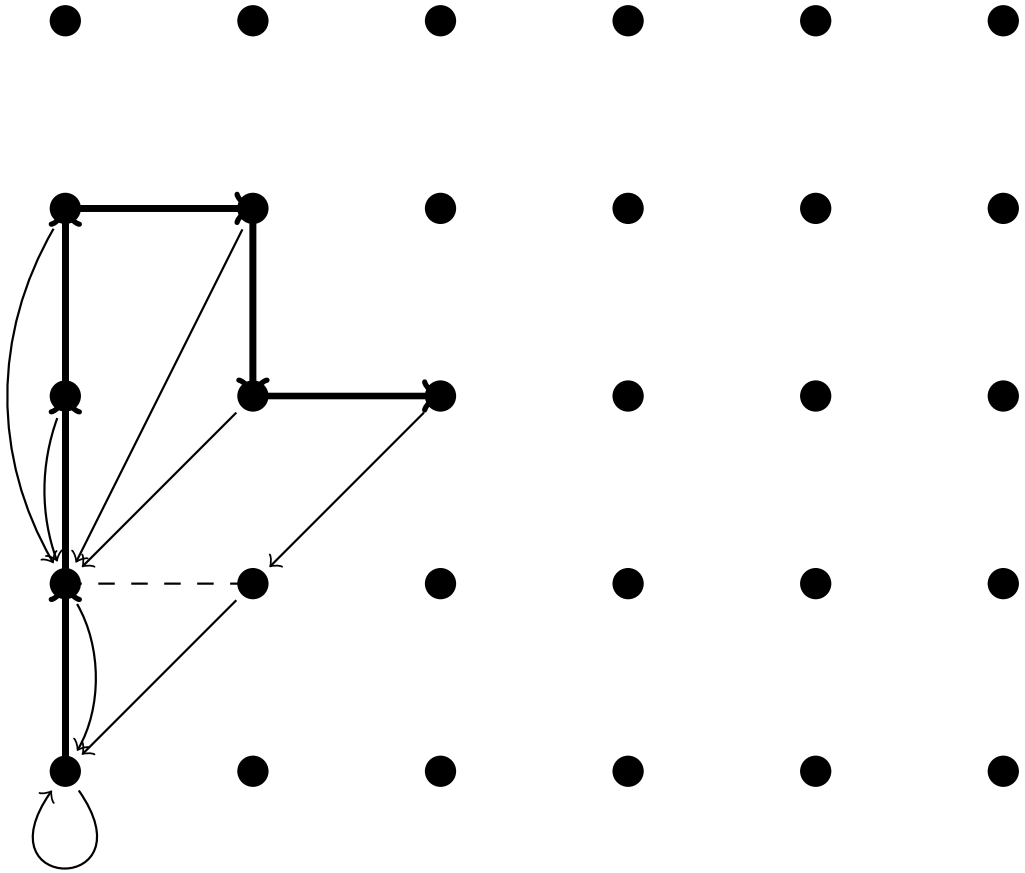
Exemple



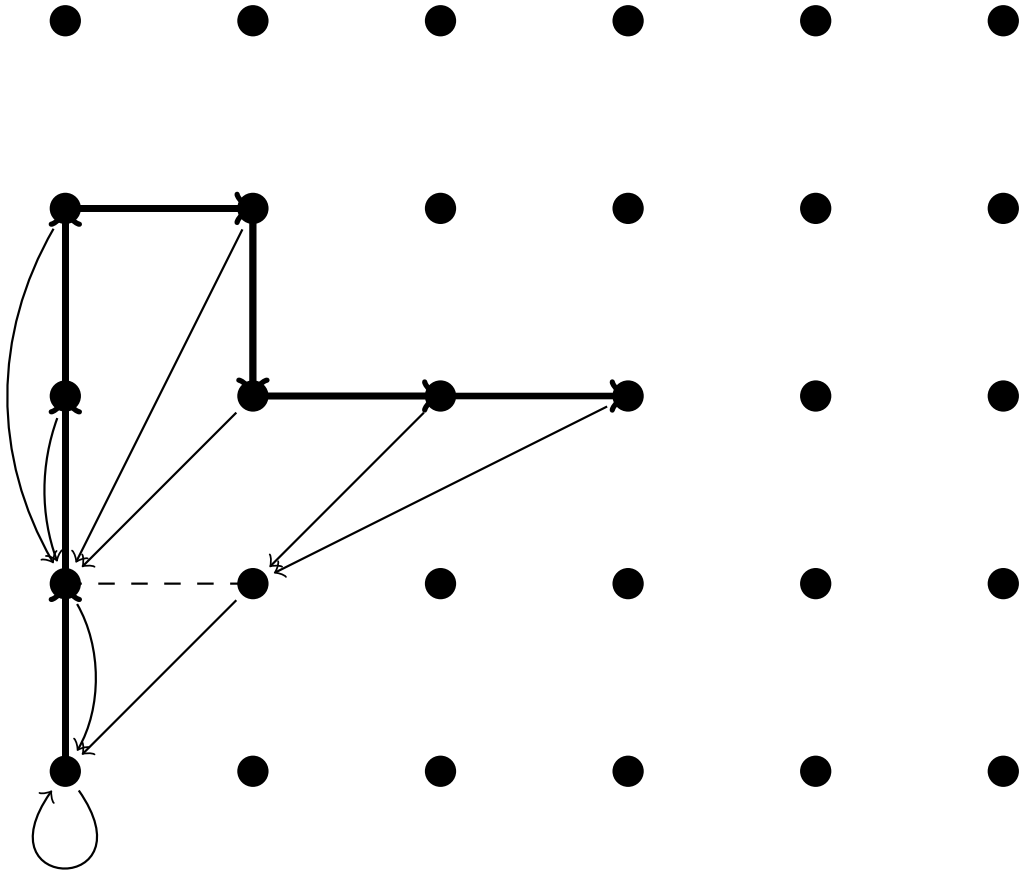
Exemple



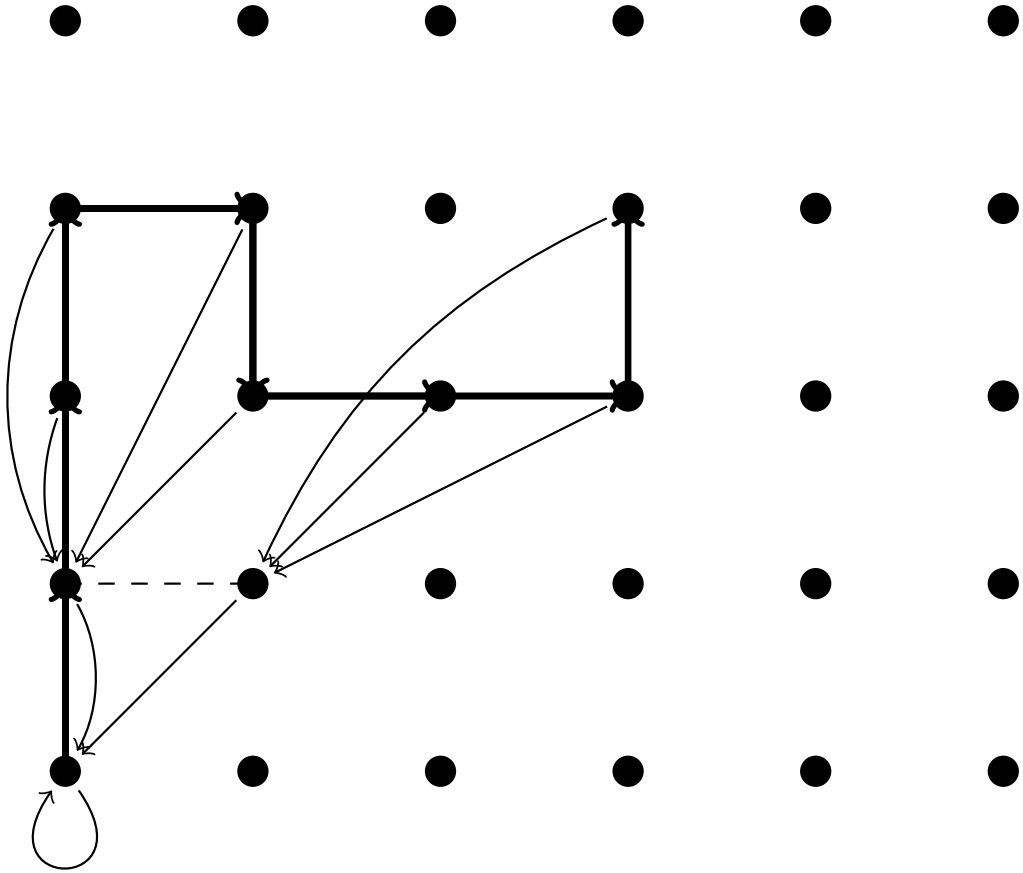
Exemple



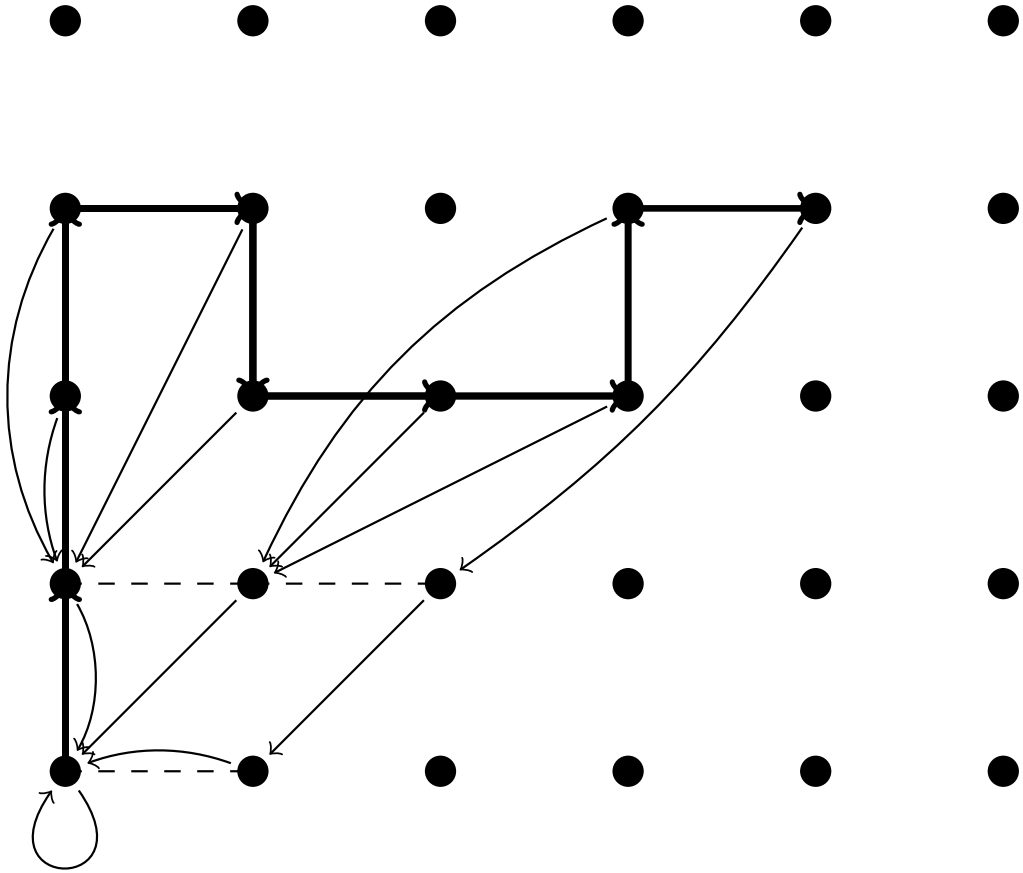
Exemple



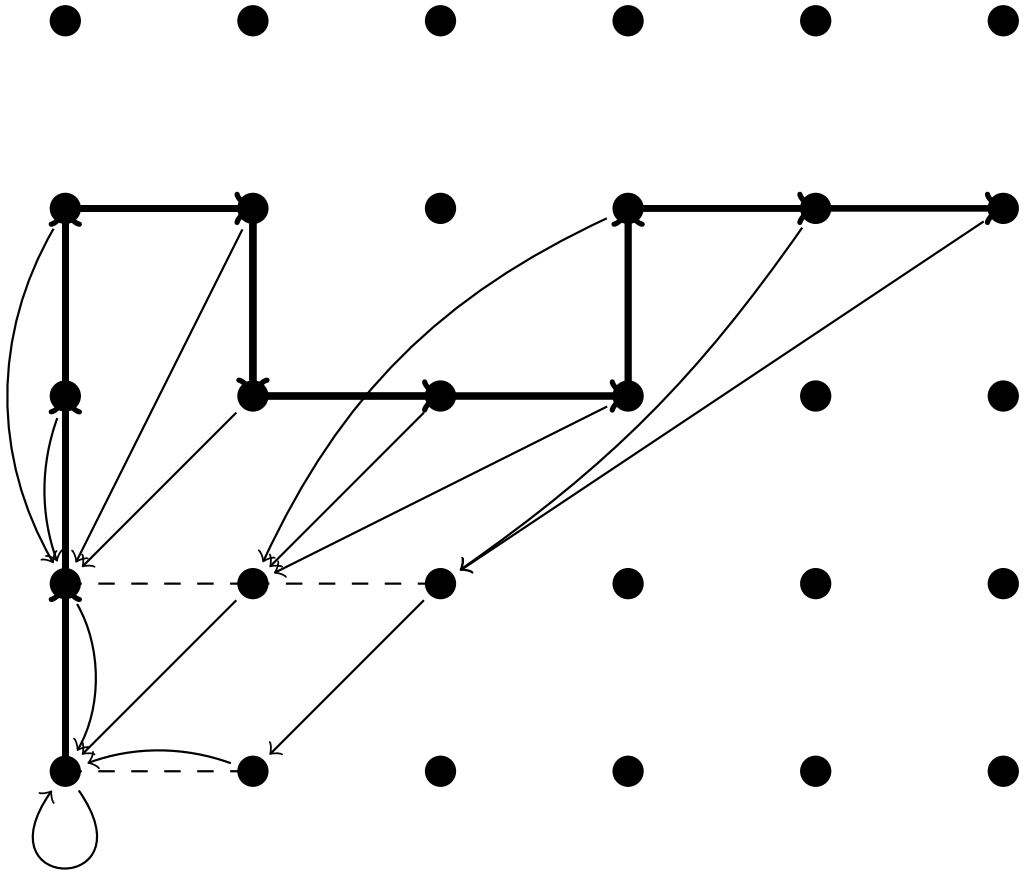
Exemple



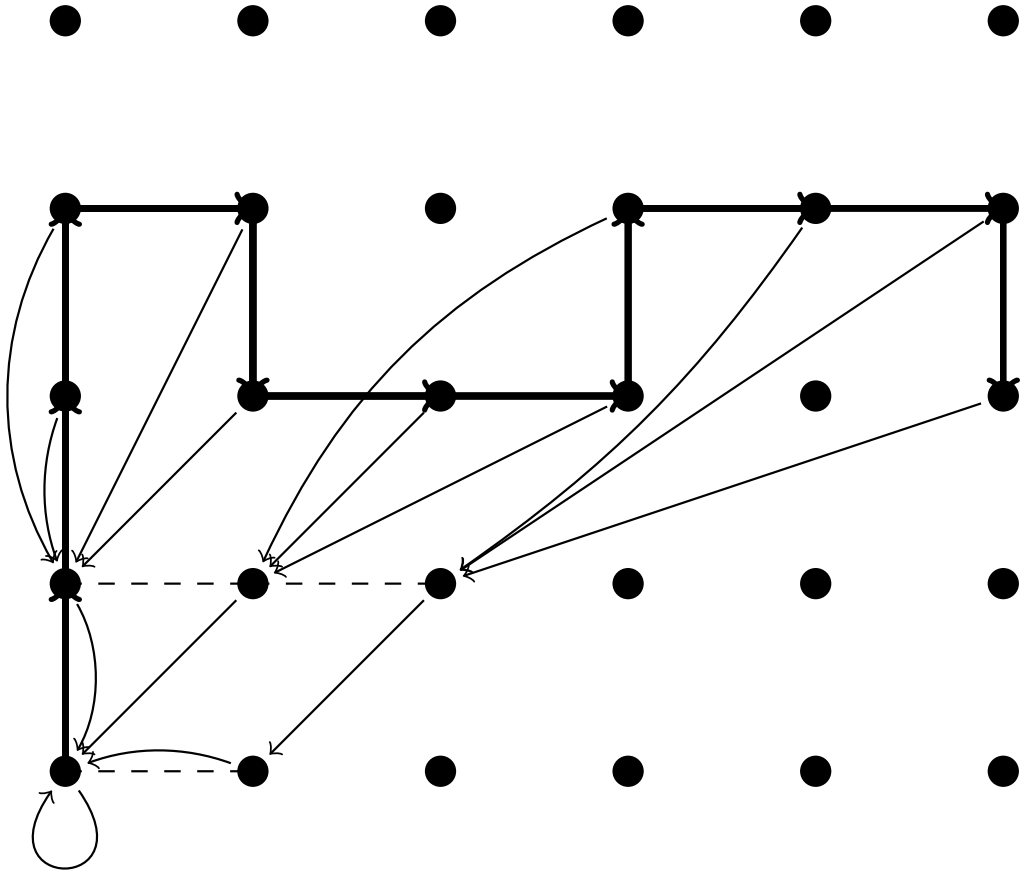
Exemple



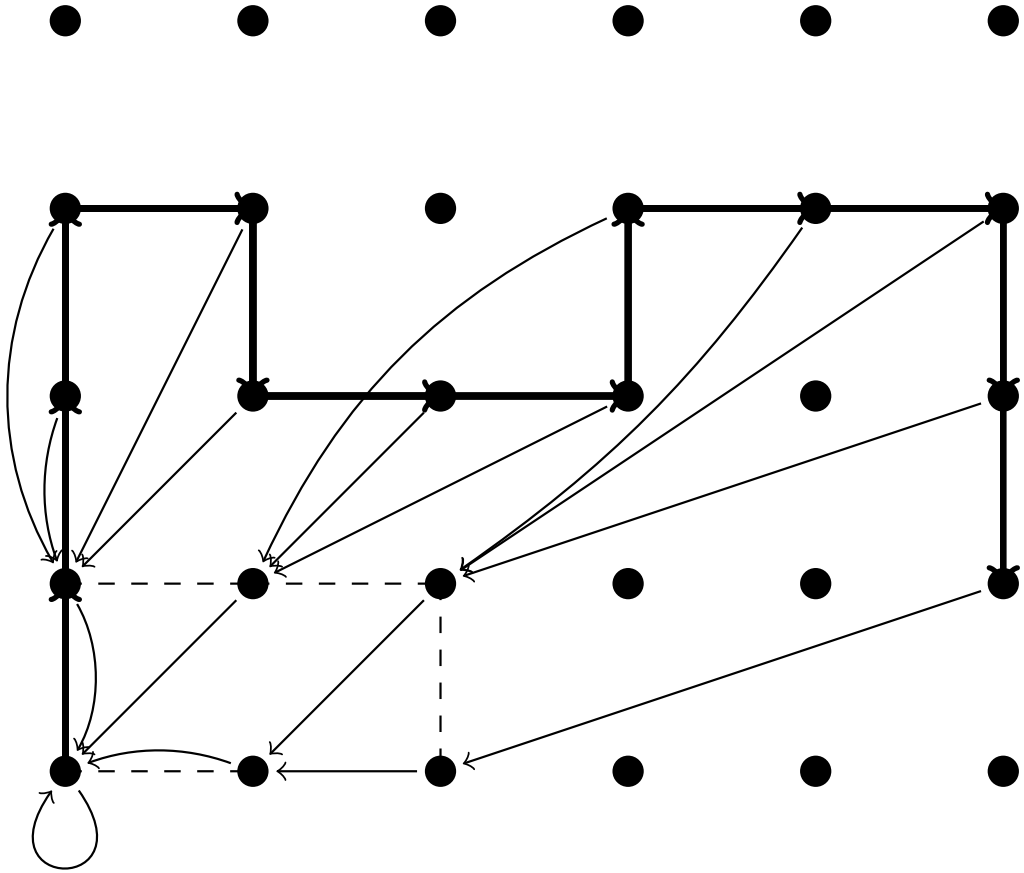
Exemple



Exemple



Exemple



Pseudocode (1/3)

```
1:  fonction DiscretePathGraph(): discrete path graph
2:      u ← Node()
3:      u.point ← (0, 0)
4:      u.parent ← u
5:      u.visited ← vrai
6:      G.current_node ← u
7:      retourner G
8:  fin fonction
9:
10: procedure Move(G : radix tree, d : direction)
11:     next_node ← Neighbor(G.current_node, d)
12:             ▷ Do something with next_node and current_node
13:     G.current_node ← next_node
14: fin procedure
```

Pseudocode (2/3)

```
1:  fonction NEIGHBOR(u: node, d: direction): node
2:      si u.neighbor[d] is not defined alors
3:          p ← u.point
4:          p' ← p + d
5:          si parent (p) = parent (p') alors
6:              v ← CHILD(u.parent, d)
7:          sinon
8:              d' ← parents_direction(p, p', d)
9:              v ← CHILD(NEIGHBOR(u.parent, d'), d)
10:         fin si
11:         u.neighbor[d] ← v
12:         v.neighbor[−d] ← u
13:     fin si
14:     retourner u.neighbor[d]
15: fin fonction
```

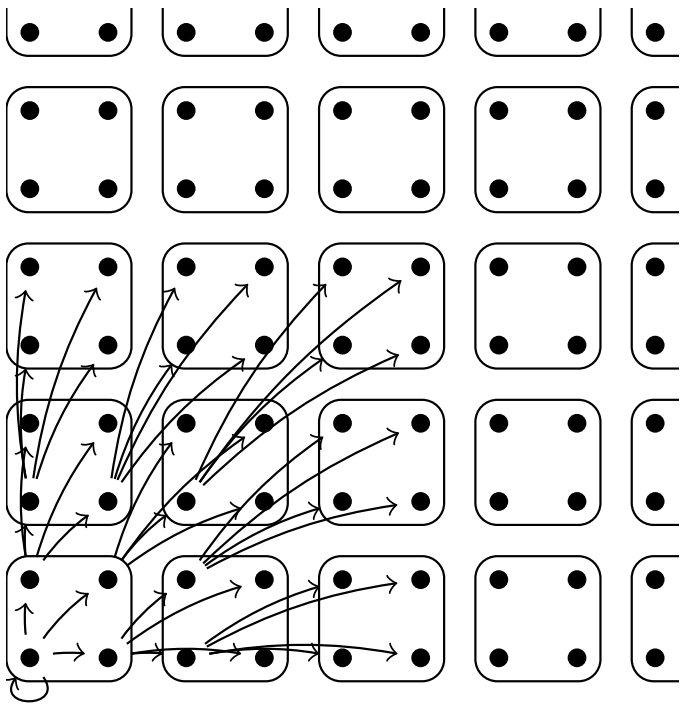
Pseudocode (3/3)

```
1:  fonction CHILD(u : node, d: direction): node
2:      si u.child[i] is not defined alors
3:          v ← NODE()
4:          i ← child_index(d)
5:          v.point ← child_coordinates(u.point, d)
6:          u.child[i] ← v
7:          v.parent ← u
8:          v.visited ← faux
9:      fin si
10:     retourner u.child[i]
11: fin fonction
```

The lemme des 4 pas

Lemme (Brlek, Koskas, Provençal, 2011)

Soit γ un chemin discret de **longueur** 4 et p_i le i-e point de $\mathbb{N}^2$ visité par γ, pour $0 \leq i \leq 4$. Alors il existe au moins deux indices distincts j et j' tels que p_j et $p_{j'}$ sont **frères**.



Complexité

- Proportionnelle au **nombre de sommets** dans le graphe, puisque le nombre de **voisins** et d'**arcs/arêtes** est borné par une **constante**;
- Mais,

$$\text{points}\,(G) = \left\{\text{points}\,(\text{parent}^{\,i}(p)) \;\middle|\; 0 \leq i \leq h, p \in \text{points}\,(\gamma)\right\}.$$

où h est la **hauteur** de l'arbre.

- Par conséquent,

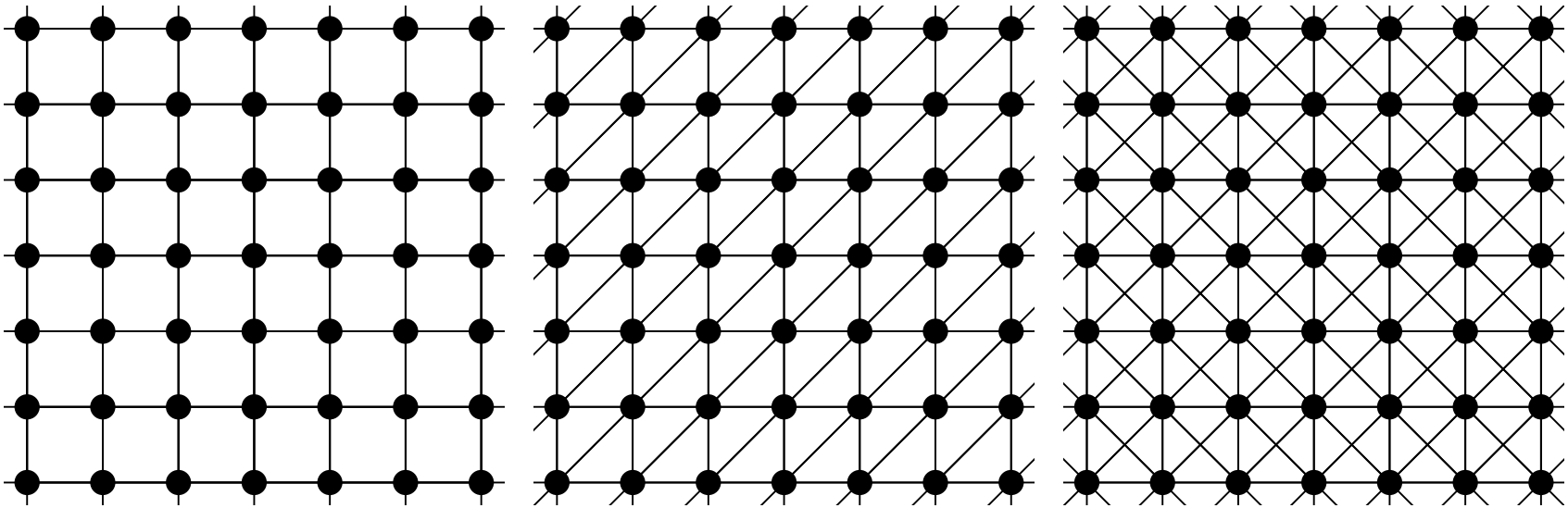
$$|\text{points}\,(\text{parent}^{\,i}(p))| \leq \left(\frac{4}{5}\right)^i |\text{points}\,(p)|$$

et

$$|\text{points}\,(G)| = \sum_{p \in \text{points}\,(\gamma)} \sum_{i=0}^{h} |\text{points}\,(\text{parent}^{\,i}(p))| \leq \sum_{i=0}^{\infty} n \left(\frac{4}{5}\right)^i = 5n = \mathcal{O}(n).$$

- La **complexité amortie** de MOVE() est $\mathcal{O}(n)/n = \mathcal{O}(1)$.

Autres grilles (1/3)

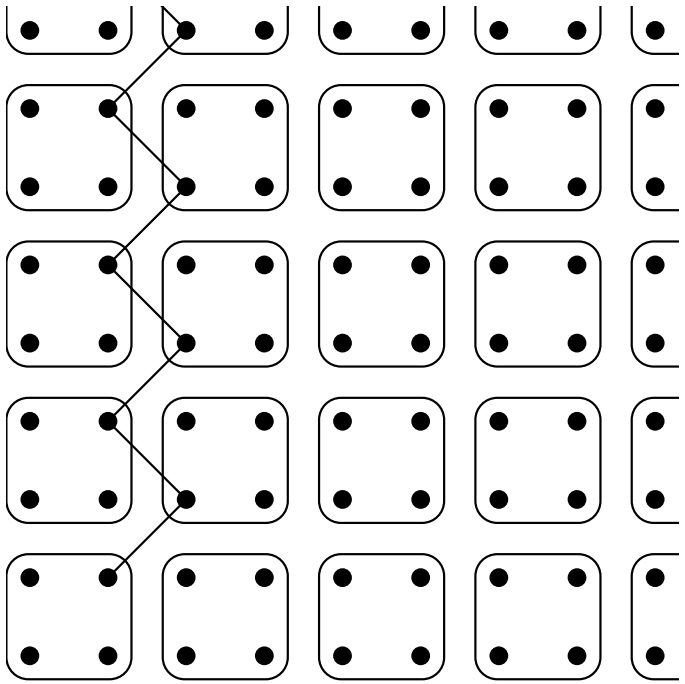


(a) (b) (c)

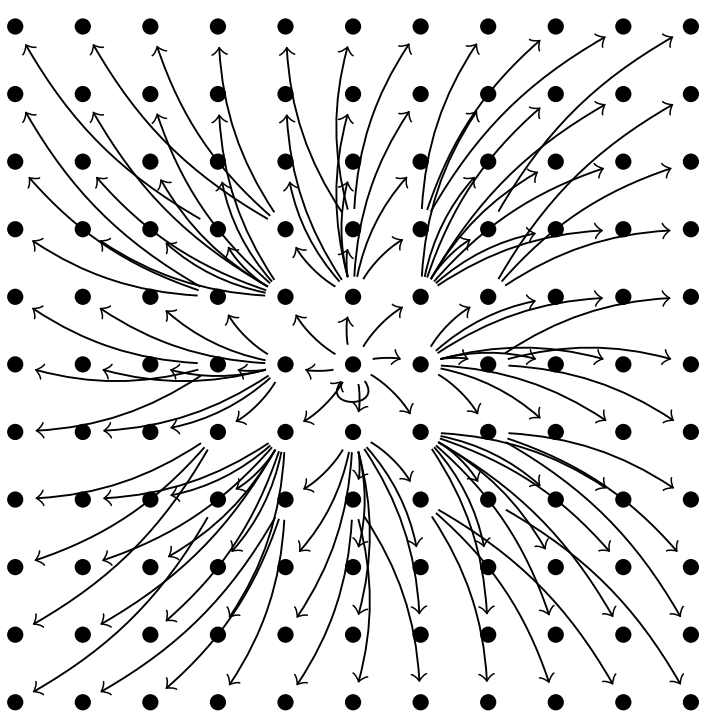
- (a) 4-régulier;
- (b) 6-régulier;
- (c) 8-régulier (non planaire).

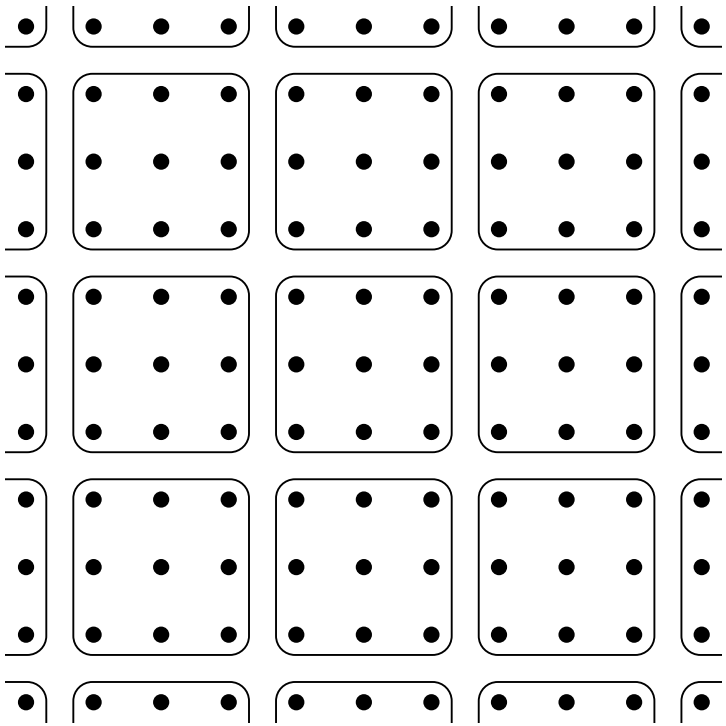
Autres grilles (2/3)

- Le lemme des 4 pas **n'est plus valide**!



Autres grilles (3/3)





- Il suffit de modifier la **relation radix**:

$$(x, y) = \left(\text{trunc}\left(\frac{x' + \text{sign}(x')}{3} \right), \text{trunc}\left(\frac{y' + \text{sign}(y')}{3} \right) \right),$$

où $\text{trunc}(z) = \text{sign}(z)\lfloor |z| \rfloor$ est la **troncature** de z.

Dernière modification

Lemme (B.M. et Marcotte, 2016)

Soit $p = (x, y), p' = (x', y') \in \mathbb{Z}^2$ deux voisins non frères tels que $p' = p + d$, où d est un vecteur élémentaire, et parents_direction(p, p') le **vector** vérifiant

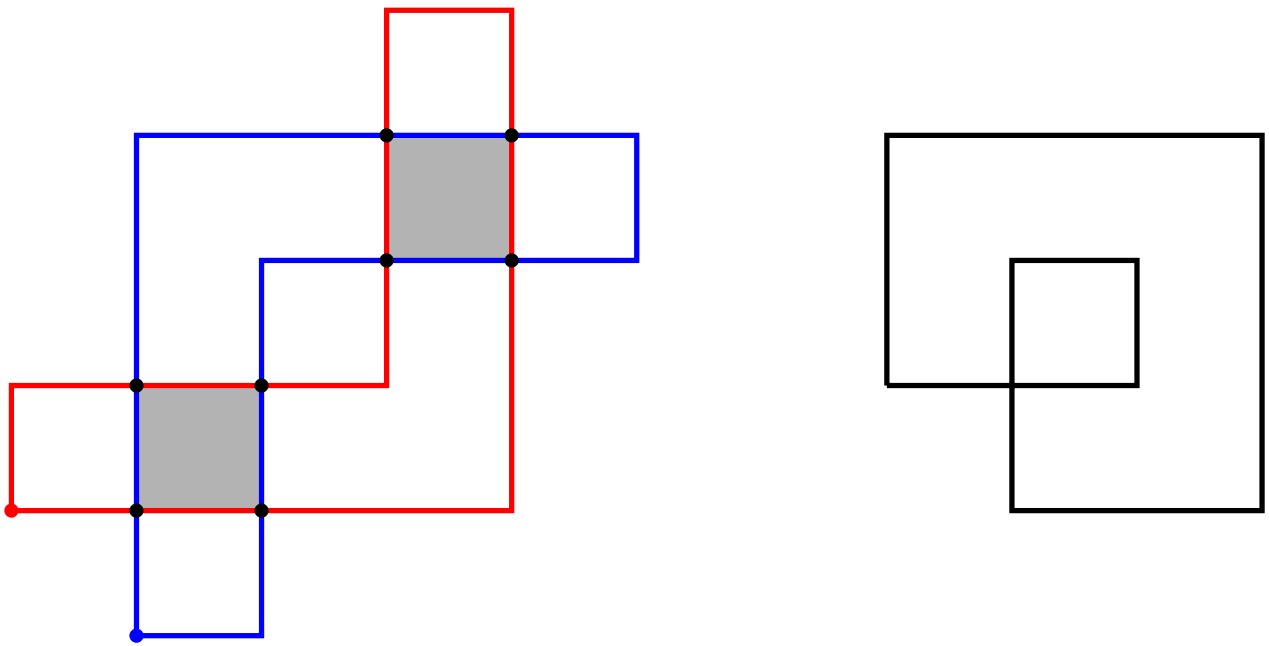
$$\text{parent}\,(p') = \text{parent}\,(p) + \text{parents\_direction}(p, p').$$

Alors

$$\text{parents\_direction}(p, p') = \begin{cases} (0, y' - y), & \text{si } (x + x') \bmod 3 \neq 0; \\ (x' - x, 0), & \text{si } (y + y') \bmod 3 \neq 0; \\ (x' - x, y' - y), & \text{sinon.} \end{cases}$$

Autres applications

- D'autres **opérations** sur les **régions discrètes** peuvent être déduites de cet algorithme avec complexités **linéaires** (Brlek, B. M., Tremblay, 2014, 2016);
- Par exemple, l'**intersection**, la **réunion**, l'**enveloppe externe**, etc.



Problèmes ouverts

- Quelles autres **opérations** peuvent être déduites de cette **structure de données**?
- Quelles **grilles** supportent cette approche (régulière, semi-régulière, non-régulière, apériodique, etc.)?
- Est-ce qu'il existe une **fonction potentielle** pour cette structure de données permettant de refaire l'**analyse de complexité amortie** pour l'opération MOVE()?

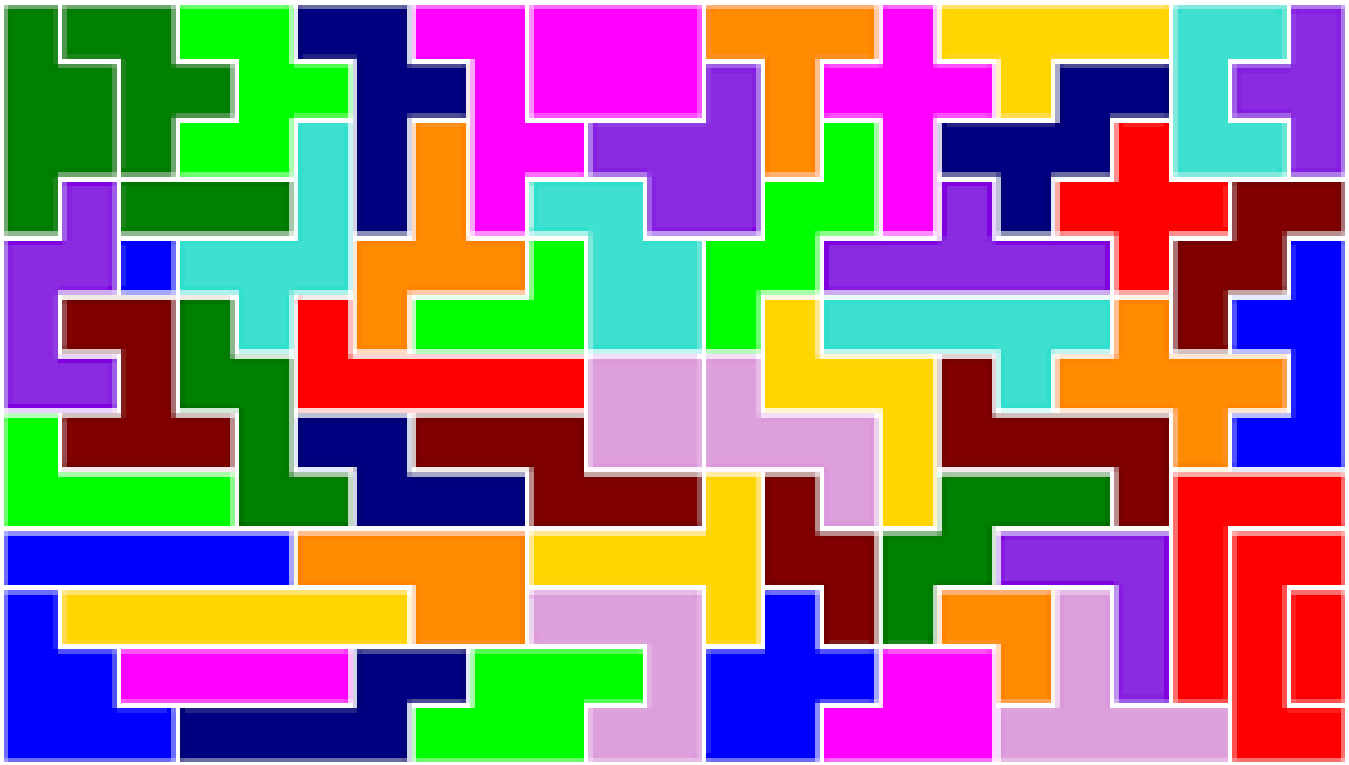
Plan

1. Chemins discrets
   - Représentation
   - Problème

2. Arbres pleinement feuillus
   - Polyominos
   - Polycubes
   - Graphes
   - Problèmes ouverts

Plan

Polyominos (1/2)

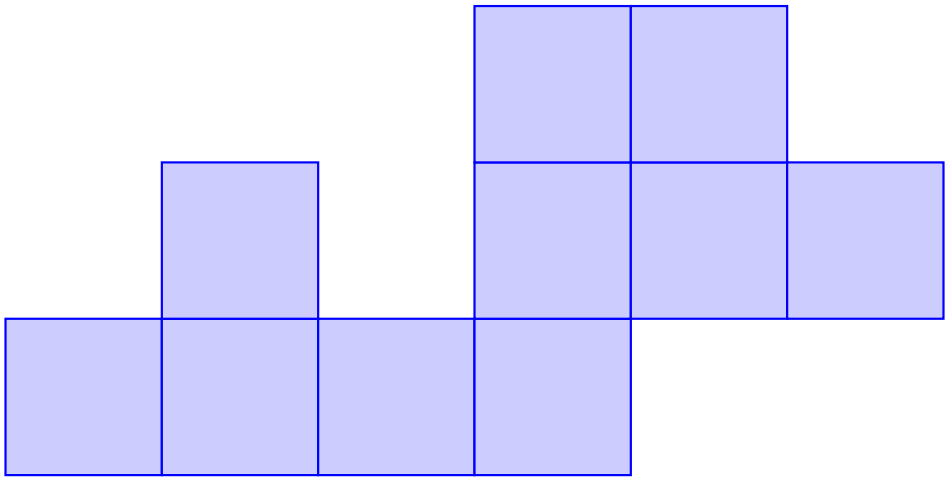


(source: David J. Goodger)

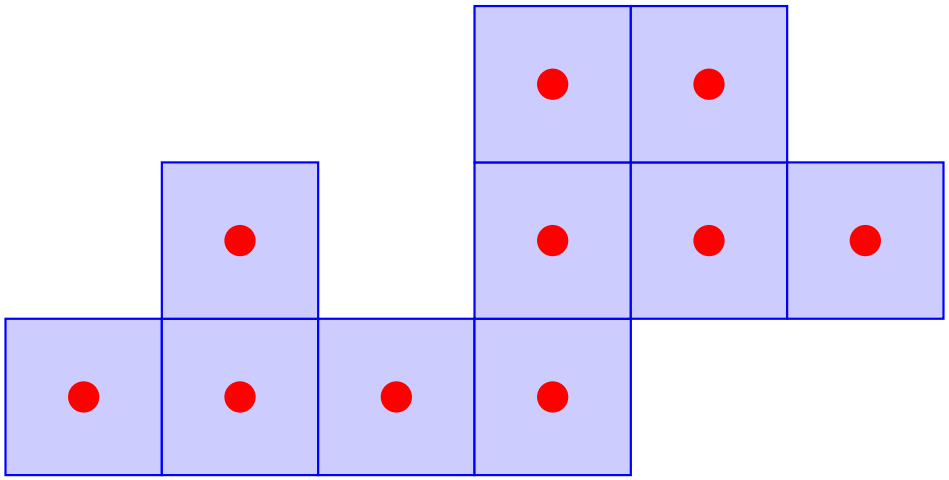
Polyominos (2/2)

- Ces objets ont beaucoup été étudiés en **combinatoire** et en **théorie des jeux**;
- Plusieurs problèmes **difficiles** :
  - Combien y a-t-il de **polyominos** de n cellules? **Non connu pour** $n \geq 57$;
  - Est-il possible de **paver** le plan euclidien à l'aide d'un ensemble de **tuiles donné**? **Indécidable en général**.
  - Est-il possible de **paver** une **région finie donnée** à l'aide d'un ensemble de tuiles donné? **NP-complet**.

Derrière chaque polyomino, il y a un graphe

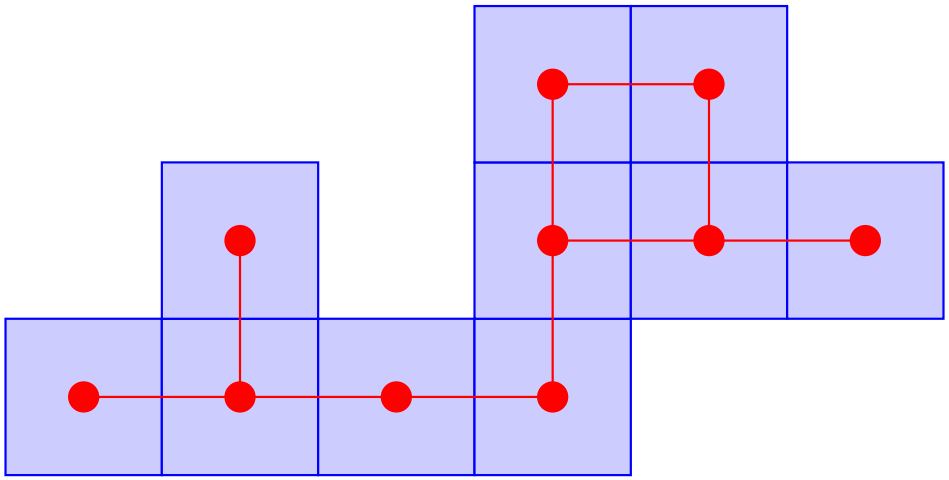


Derrière chaque polyomino, il y a un graphe



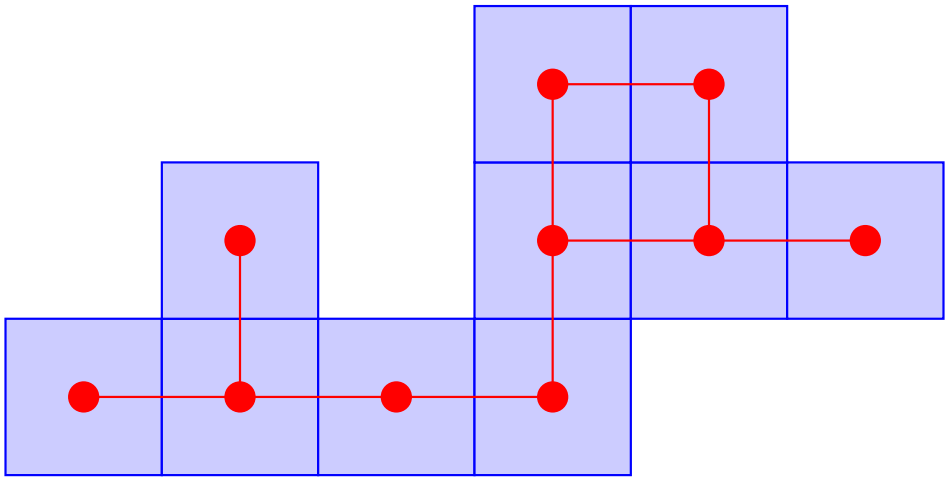
- Chaque **cellule** est un **sommet**;

Derrière chaque polyomino, il y a un graphe



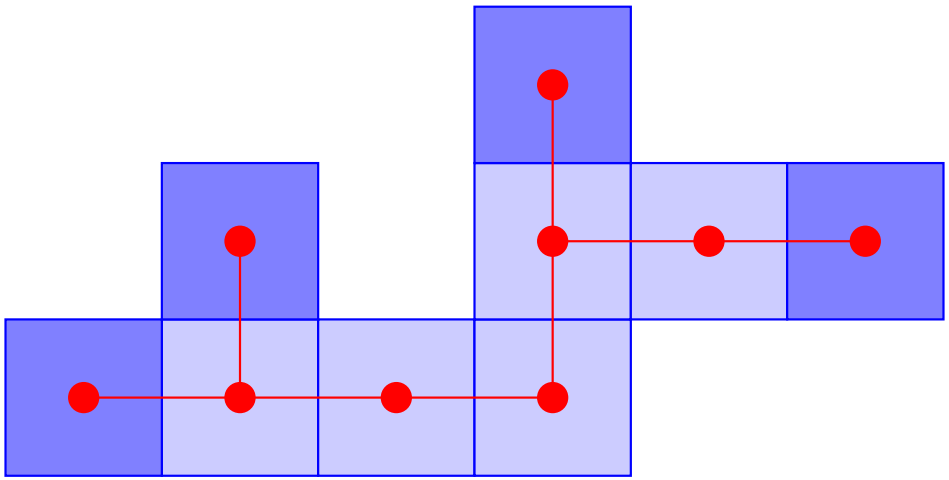
- Chaque **cellule** est un **sommet**;
- Il y a une **arête** entre deux cellules si elles partagent un **côté**;

Derrière chaque polyomino, il y a un graphe



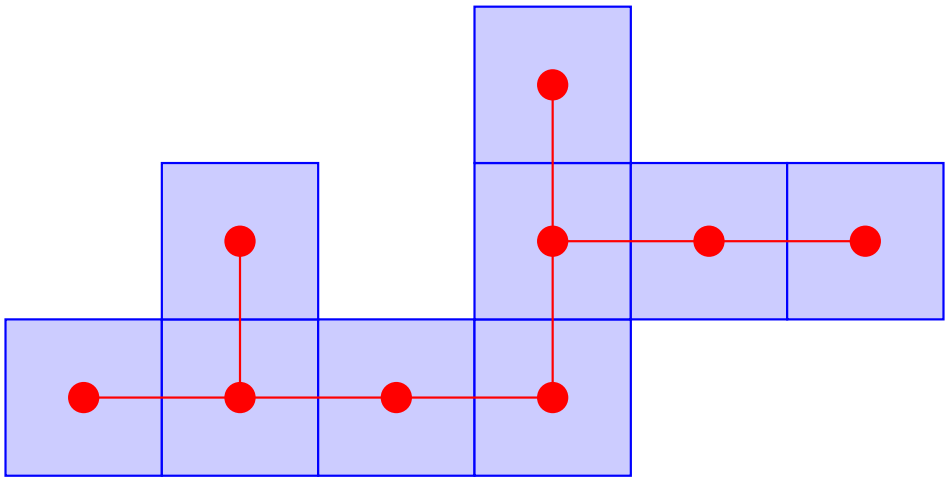
- Chaque **cellule** est un **sommet**;
- Il y a une **arête** entre deux cellules si elles partagent un **côté**;
- Ce graphe peut posséder des **cycles**.

Polyominos-arbres



- Si le graphe est **acyclique**, le polyomino est appelé **polyomino-arbre**.
- Le polyomino-arbre ci-haut possède 4 **feuilles**.

Nombre maximal de feuilles?



Soit $L_2(n)$ le nombre maximum de feuilles qui peut être réalisé par un polyomino de n **cellules**.

Définition

Soit T un polyomino-arbre de n cellules et ℓ feuilles. On dit que T est **pleinement feuillu** si $\ell = L_2(n)$.

Nombre maximum de feuilles

Théorème

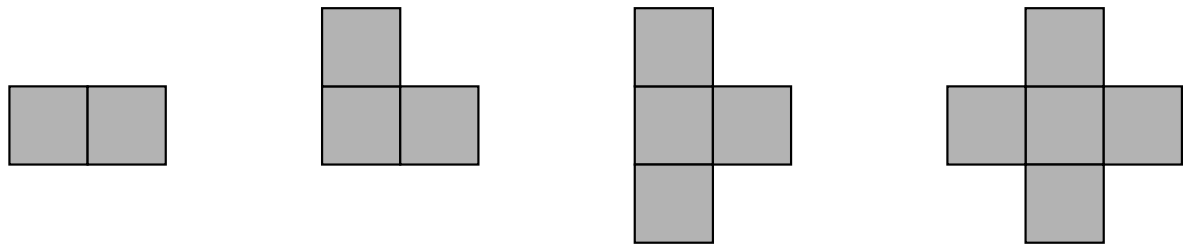
[B. M., Goupil, De Carufel, Samson; 2016] Pour tout entier $n \geq 1$, soit

$$\ell_2(n) = \begin{cases} 0, & \text{si } n = 1; \\ 2, & \text{si } n = 2; \\ n-1, & \text{si } n = 3, 4, 5; \\ \ell_2(n-4) + 2, & \text{si } n \geq 6. \end{cases}$$

Alors $L_2 = \ell_2$.

Idée de la démonstration $L_2 \geq \ell_2$

- Le cas $n = 1$ est trivial;
- On construit ensuite une famille par **récurrence sur** $n \geq 2$.
- **Cas de base**:

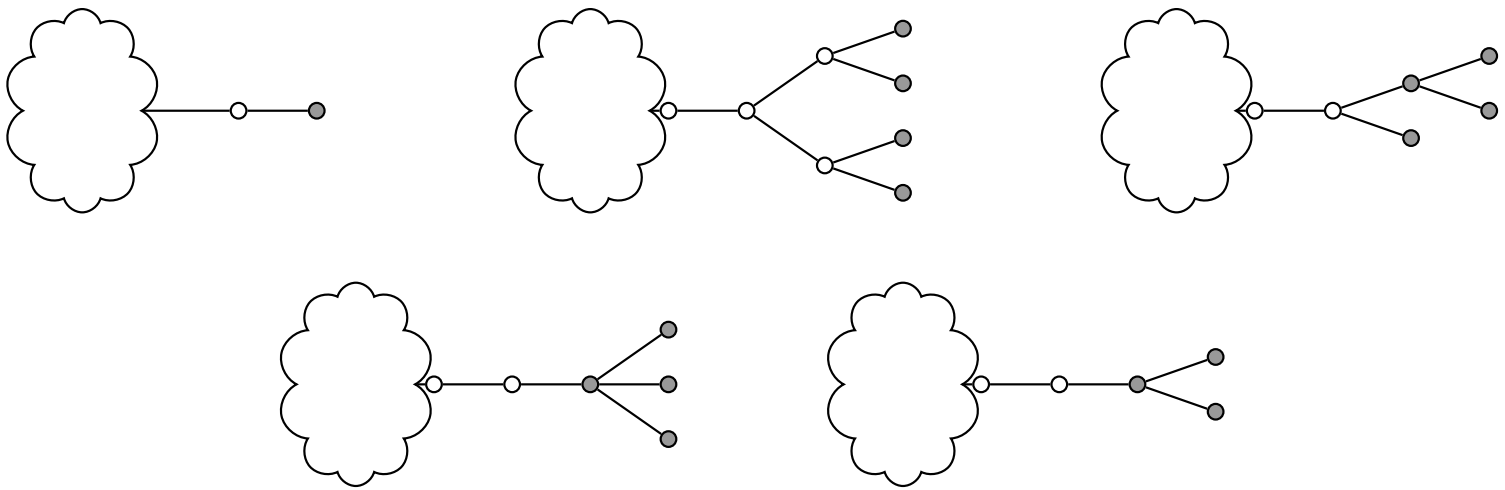


$n = 2$	$n = 3$	$n = 4$	$n = 5$
$\ell = 2$	$\ell = 2$	$\ell = 3$	$\ell = 4$

- **Récurrence**: On construit un polycube T_{n+4} en **"annexant"** le polyomino $n = 4$ à la droite de T_n, ce qui nous fait **"gagner" 3 feuilles**, mais en **"perdre" une**.

Idée de la démonstration $L_2 \leq \ell_2$

- On procède par **contre-exemple minimal**;
- Supposons qu'il existe un arbre de $> \ell_2(n)$ feuilles.
- On peut choisir n **minimal**.
- Alors on peut construire un **contre-example** de **taille** $< n$



Plan

Polycubes

- L'équivalent d'un polyomino en 3D est appelé **polycube**;
- Chaque cellule possède au plus 6 **voisins** plutôt que 4;
- La même question se pose: quel est le **nombre maximal** de feuilles $L_3(n)$ qui peut être réalisé avec un polycube-arbre de n **cellules**?

Nombre maximum de feuilles (3D)

Théorème

[B. M., Goupil, De Carufel, Samson; 2016] Pour tout entier $n \geq 1$, soit

$$\ell_3(n) = \begin{cases} 0, & \text{si } n = 1; \\ f_3(n) + 1, & \text{si } n = 6, 7, 13, 19, 25; \\ f_3(n), & \text{si } 0 \leq n \leq 40 \text{ et } n \neq 1, 6, 7, 13, 19, 25; \\ f_3(n - 41) + 28, & \text{si } 41 \leq n \leq 81; \\ \ell_3(n - 41) + 28, & \text{si } n \geq 82. \end{cases}$$

où

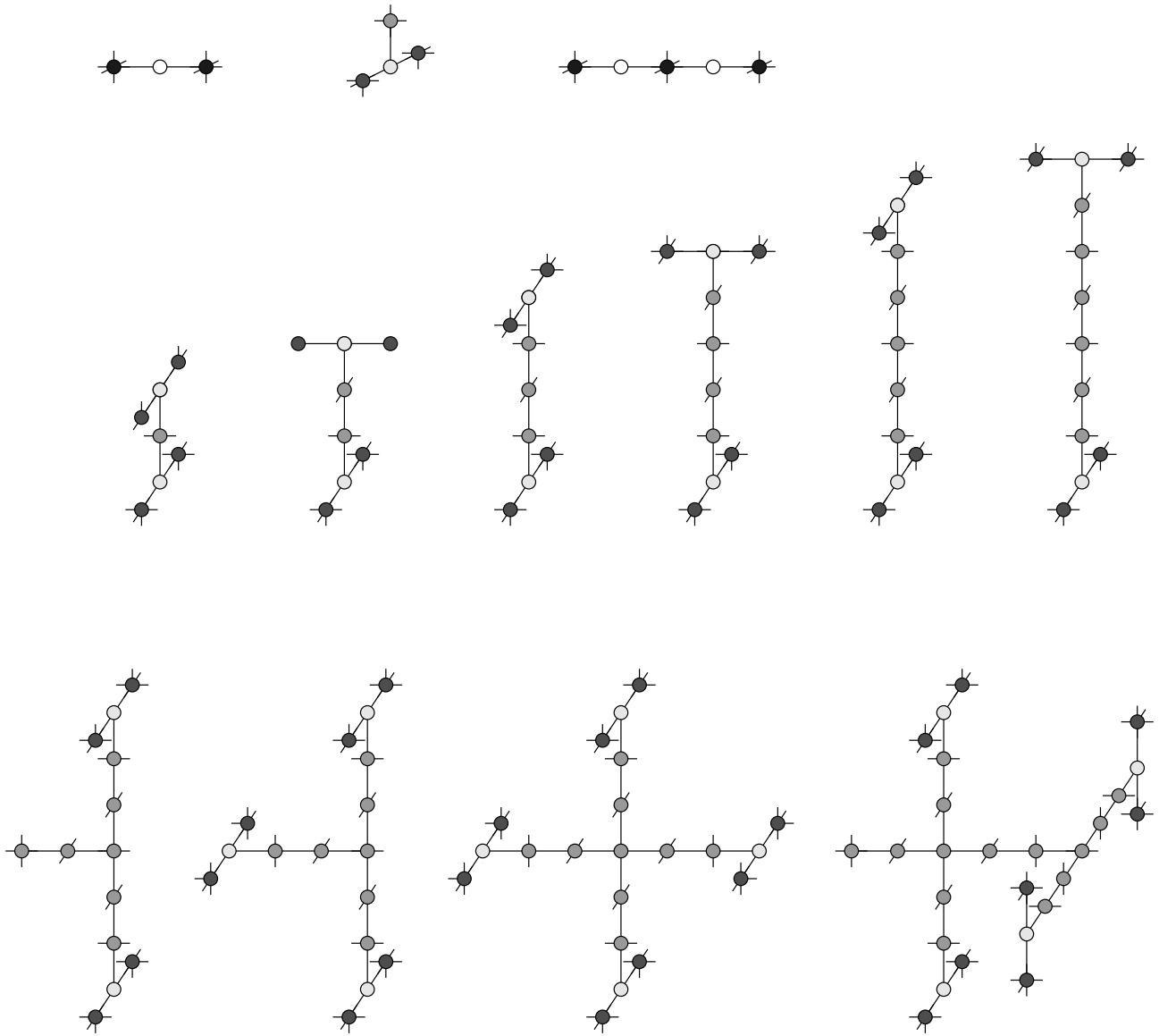
$$f_3(n) = \begin{cases} \lfloor (2n + 2)/3 \rfloor, & \text{si } 0 \leq n \leq 11; \\ \lfloor (2n + 3)/3 \rfloor, & \text{si } 12 \leq n \leq 26; \\ \lfloor (2n + 4)/3 \rfloor, & \text{si } 27 \leq n \leq 40. \end{cases}$$

Alors $L_3 = \ell_3$.

Idée de la démonstration

- Similaire à celle pour le cas des **polyominos**;
- On démontre que $L_3 \geq \ell_3$ en **construisant** une famille de polycubes-arbres (**facile**);
- On démontre que $L_3 \leq \ell_3$ par **contre-exemple minimal**, avec deux **difficultés additionnelles**:
  - **Explosion combinatoire** du nombre de cas à **examiner** (il faut recourir à un **programme informatique**);
  - L'**élagage simple** pour construire des plus petits **contre-exemples** ne suffit pas (il faut introduire la notion de **substitution**).

Exemples (voir https://youtu.be/b7Wf-03449o)



Plan

Sous-arbres induits feuillus (SAIF)

On peut généraliser le problème de la façon suivante :

Problème SAIF

Étant donné un graphe non orienté G et deux entiers n, ℓ, est-ce qu'il existe un **sous-arbre induit** de G de n **cellules** et de ℓ **feuilles**?

Définition

Soit $G = (V, E)$ un graphe non orienté et $U \subseteq V$. Le **sous-graphe induit par** U, noté $G[U]$, est défini par

$$G[U] = (U, E \cap \mathcal{P}_2(U)).$$

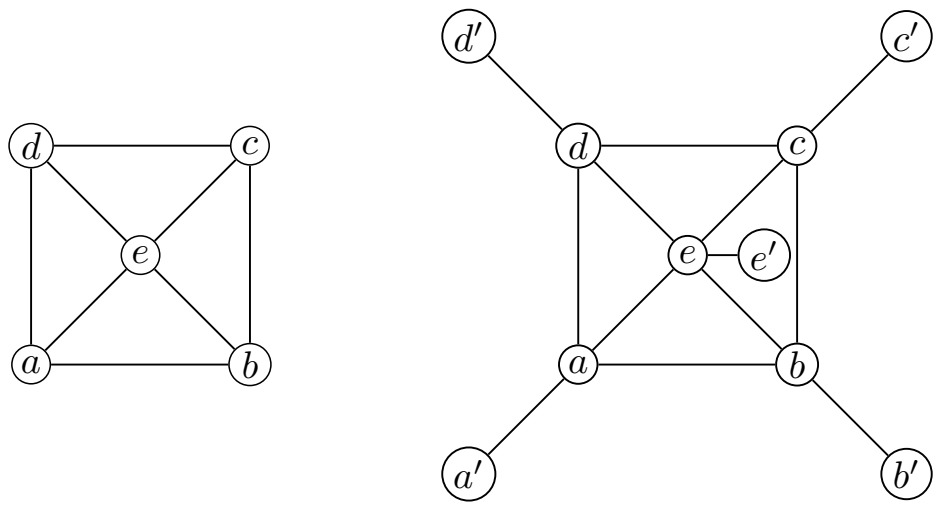
SAIF est NP-complet

Théorème

[B. M., Goupil, De Carufel, Vandomme; 2017] Le problème SAIF est **NP-complet** sur les graphes non orientés.

Proof

Réduction à partir du **problème du sous-arbre induit** (Erdös, 1986).



G H

Graphes à largeur arborescente bornée

Théorème

[B. M., Goupil, De Carufel, Vandomme; 2017] Le problème SAIF est **polynomial** lorsqu'on se restreint aux graphes à **largeur arborescente bornée**, en particulier les **arbres**.

Idée de la démonstration

- On peut trouver une **décomposition arborescente** d'un graphe en **temps polynomial**;
- Le problème SAIF s'exprime comme une formule **logique monadique du deuxième ordre**;
- En vertu d'un résultat de **Courcelle** (1990), un algorithme polyomial existe.

Arbres

Théorème

[B. M., Goupil, De Carufel, Vandomme; 2017] Soit $T = (V, E)$ un arbre, avec $n = |V|$. Alors L_T peut être calculé en temps $\Theta(n^3)$ et en espace $\Theta(n^2)$.

Idée de la démonstration

- **Programmation dynamique**;
- Pour chaque **orientation d'une arête** (u, v) de T, on calcule une **fonction partielle** $L_{u \to v}$ des **feuilles** vers le **centre** du graphe;
- Puis on calcule L_T en examinant **chaque arête** et en calculant la somme $L_{u \to v}$ and $L_{v \to u}$.

Plan

Prochaine étape?

- Nous savons calculer L_2 (polyominos), L_3 (polycubes) et L_T (si T est un arbre);
- Y a-t-il d'autres **familles intéressantes de graphes** pour lesquelles le problème est **polynomial**?
- Que se passe-t-il sur d'**autres réseaux** (pavages de Penrose, pavages hyperboliques, etc.)?
- Qu'en est-il de **compter** et d'**énumérer**, pour chaque n, le nombre de polyominos-arbres, polycubes-arbres **pleinement feuillus**?