Carte routière

- Quelle structure ?

Carte routière

- Quelle structure ? Graphe non orienté
- Sommets : ?
- Arêtes : ?

Carte routière

- Quelle structure ? Graphe non orienté
- Sommets : Villes
- Arêtes : Route directe (+ distance)

Carte routière

- Quelle structure ? Graphe non orienté
- Sommets : Villes
- Arêtes : Route directe (+ distance)
- Implémentation ?

Carte routière

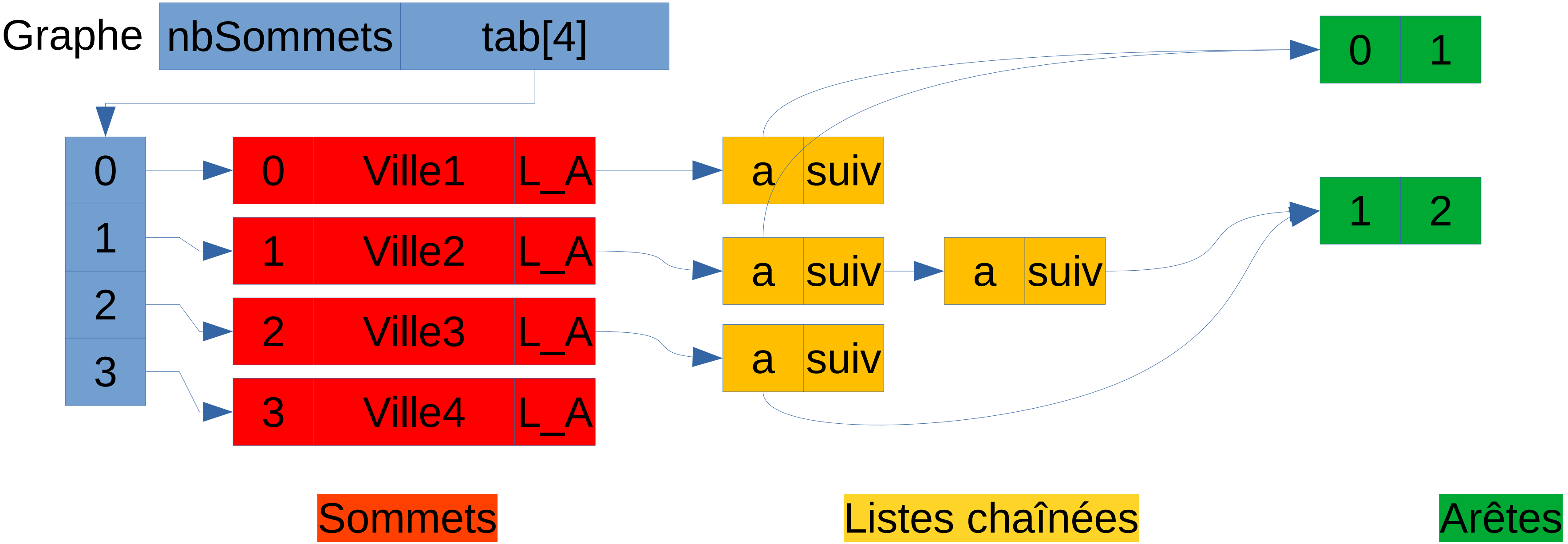
- Quelle structure ? Graphe non orienté
- Sommets : Villes
- Arêtes : Route directe (+ distance)
- Implémentation ? Matrice ou **liste d'adjacence**

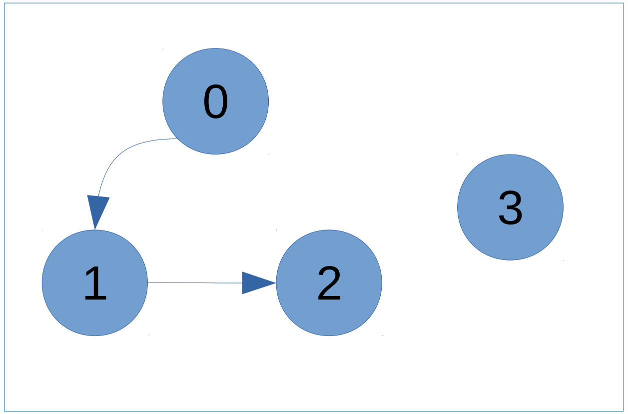
Carte routière

- Quelle structure ? Graphe non orienté
- Sommets : Villes
- Arêtes : Route directe (+ distance)
- Implémentation ? Matrice ou **liste d’adjacence**

Matrice creuse = mémoire gaspillée car beaucoup de 0, donc de ‘’cases inutiles’’

Représentation limitée aux arêtes, donc pas de mémoire gaspillée pour 2 villes non reliées directement


Graphe
nbSommets
tab[4]
0
1
2
3
0
Ville1
L_A
1
Ville2
L_A
2
Ville3
L_A
3
Ville4
L_A
a
suiv
a
suiv
a
suiv
a
suiv
0
1
1
2
Sommets
Listes chaînées
Arêtes


0
1
2
3

Complexité calcul de degré

- Matrice adjacence :
  - Degré sortant :
  - Degré entrant :
- Listes d’adjacence :
  - Degré sortant :
  - Degré entrant :

Complexité calcul de degré

- Matrice adjacence :
  - Degré sortant : $\Theta(n)$
  - Degré entrant : $\Theta(n)$
- Listes d'adjacence :
  - Degré sortant :
  - Degré entrant :

Complexité calcul de degré

- Matrice adjacence :
  - Degré sortant : $\Theta(n)$
  - Degré entrant : $\Theta(n)$
- Listes d'adjacence :
  - Degré sortant : $O(n)$?
  - Degré entrant : $O(n^2)$?

Complexité calcul de degré

- Matrice adjacence :
  - Degré sortant : $\Theta(n)$
  - Degré entrant : $\Theta(n)$
- Listes d'adjacence :
  - Degré sortant : $O(n)$?
  - Degré entrant : $O(n^2)$?

NON

Complexité calcul de degré

- Listes d’adjacence degré sortant
  - Il faut regarder plus en détails en regardant un paramètre statistique : la taille maximale d’une liste d’adjacence
  - Si on note a cette taille alors on est en O(a)
  - Graphe très dense → sommet a environ n successeurs → a proche de n → les fonctions sont comparables
  - Graphe peu dense → a petit → deuxième fonction meilleure

Complexité calcul de degré

- Listes d'adjacence degré entrant
  - $O(n^2)$ si on ne connaît pas le nombre d'arcs
  - $O(n + m)$ si on connaît m le nombre d'arcs
  - L'implémentation avec matrice est meilleure si on a $m > n$
  - En réalité il existe une implémentation des listes à peine plus coûteuse en mémoire où on peut calculer le degré sortant en $O(a)$: chaque sommet contient une liste chaînée de ses prédécesseurs

Coloration

- 3.1 : On utilise un graphe non orient´e : les sommets sont les antennes, et les arêtes sont les paires d’antennes qui sont en conflit.

Coloration

- 3.2 : La coloration de graphe est un problème NP-Complet = on ne connaît pas d’algorithme en temps polynomial pour le résoudre

Coloration

```
nbCouleurs = 0
tabCouleurs[] (initialisé avec des -1)
nbSommets = n
nbSommetsColories = 0
tabSommets[]
while(nbSommetsColories!= nbSommets) {
    Sommet* s = get_sommet_non_colorie(tabSommets)
    if(nb_voisins_colories(s) >= nbCouleurs) {
        nbCouleurs++
        s→couleur = nbCouleurs
    } else {
        s→couleur =  couleur_non_utilisee_par_voisins(s)
    }
    nbSommetsColories++
}
```

Coloration

```
nbCouleurs = 0
tabCouleurs[] (initialisé avec des -1)
nbSommets = n
nbSommetsColories = 0
tabSommets[]
while(nbSommetsColories!= nbSommets) {
    Sommet* s = get_sommet_non_colorie(tabSommets)
    if(nb_voisins_colories(s) >= nbCouleurs) {
        nbCouleurs++
        s→couleur = nbCouleurs
    } else {
        s→couleur =  couleur_non_utilisee_par_voisins(s)
    }
    nbSommetsColories++
}
```

Ne retourne pas forcément la solution optimale !

Coloration

- 3.4 : A chaque itération, on choisit un sommet et on parcourt ses voisins. Avec une matrice d’adjacence,on obtient une complexité totale en $\Theta(n^2)$, tandis qu’avec des listes d’adjacence on est en $O(n+m)$. **Quand $m \ll n^2$, les listes d’adjacences sont à préférer**.