

La théorie des graphes pour les noobs

Par Olipro

Maitre de conférence à l'université de St Pouillaux Les Marais.

1736 : premier probleme de graphe. Les ponts de Koniskbergs, par Euler (Mathématicien Suisse)
1835 : probleme de flot et coupe. La loi des circuits, par Gustav Kirshhoff (Physicien Allemand)
-> theoreme min max
1852 : probleme des 4 couleurs. Par Francis Guthrie (Mathématicien Sud-Africain)
1914 : "[tout graphe biparti régulier a un couplage parfait](#)" par König
1927 : **théorème de Menger** (ancete du theoreme de flot-max/coupe-min)
1931 : Théorème de König
1935 : Theoreme de Hall (origine de la theorie de la complexité)
1956 : theoreme de flot-max/coupe-min (programmation linéaire)+ [algorithme de Kruskal](#) (premier [algorithme glouton](#)).
1960 : Conjecture de Berge
1976 : Theoreme des 4 couleurs
2002 : Theoreme des graphes parfaits
2008 : Theoreme du graphe tres glouton, par D. Vellard.

Theoremes :

Menger :

le nombre minimum d'arêtes dont la suppression déconnecte deux sommets s et t est égal au nombre maximum de chemins arête-disjoints reliant s et t.

Konig :

Pour tout graphe biparti G, la cardinalité maximum d'un couplage de G est égale à la cardinalité minimum d'un transversal de G.

Hall :

Un graphe biparti $G=(U,V;E)$ admet un couplage parfait si et seulement si pour tout sous-ensemble X de U (de V, respectivement), le nombre de sommets de V (de U, respectivement) adjacents à un sommet de X est supérieur ou égal à la cardinalité de X.

Theoreme des 4 couleurs :

il est possible, en n'utilisant que quatre couleurs différentes, de colorer n'importe quelle carte découpée en [régions connexes](#), de sorte que deux régions *adjacentes* (idem *limitrophes*) reçoivent toujours deux couleurs distinctes.

Theoreme des graphes parfaits :

Théorème *faible* des graphes parfaits :

Un graphe est parfait si et seulement si son [complémentaire](#) est parfait.

Théorème *fort* des graphes parfaits :

Un graphe est parfait si et seulement si ni lui ni son complémentaire ne contiennent de [cycle impair](#) induit de longueur au moins cinq.

Théoreme des graphes très gloutons :

La quantité d'éléments que peut absorber un graphe est inversement proportionnelle à la qualité informative que ce graphe délivre aux sous-graphes.

Algos et notions:

algo de Malgrange (algo du rang ?) :

probleme : trouver les composantes fortement connexes.

consiste à faire l'intersection de tous les successeur et predecesseurs afin de trouver les composantes fortement connexes (= des circuits quoi...) Apres on relie tous les circuits trouvés entre eux comme si c'était un super nouveau graphe et youplalol. Si on a deux circuits avec deus fleches dans deux sens, c'est qu'il y a une co(q)uille.

Algo de dijkstra :

Bien savoir manier le super tableau comme au DS de reseau.

Graphe eulerien :

nombre pair d'aretes

graphe hamiltonien :

aucun sommet de degres 1

Ford fulkerson : (avec schema)

Probleme du flot maximal.

On a un depart et une destination

Etape 1 : saturer comme un porc au hasard entre le depart et l'arrivée

Etape 2 : faire le nouveau graphe des reseaux residuels.

(Ex : si on avait $A \xrightarrow{3/5} B$ avec 3/5 sur la fleche, on a alors deux fleches $A \xrightarrow{2} B$ (avec 2) et $A \xleftarrow{3} B$ (avec 3))

On a un nouveau graphe bien crade avec des fleches partout.

While(y a moyen de saturer)

{

Etape 3 : DEPUIS CE NOUVEAU GRAPHE : saturer comme un porc à nouveau.

Etape 4 : faire le nouveau graphe des reseaux residuels.

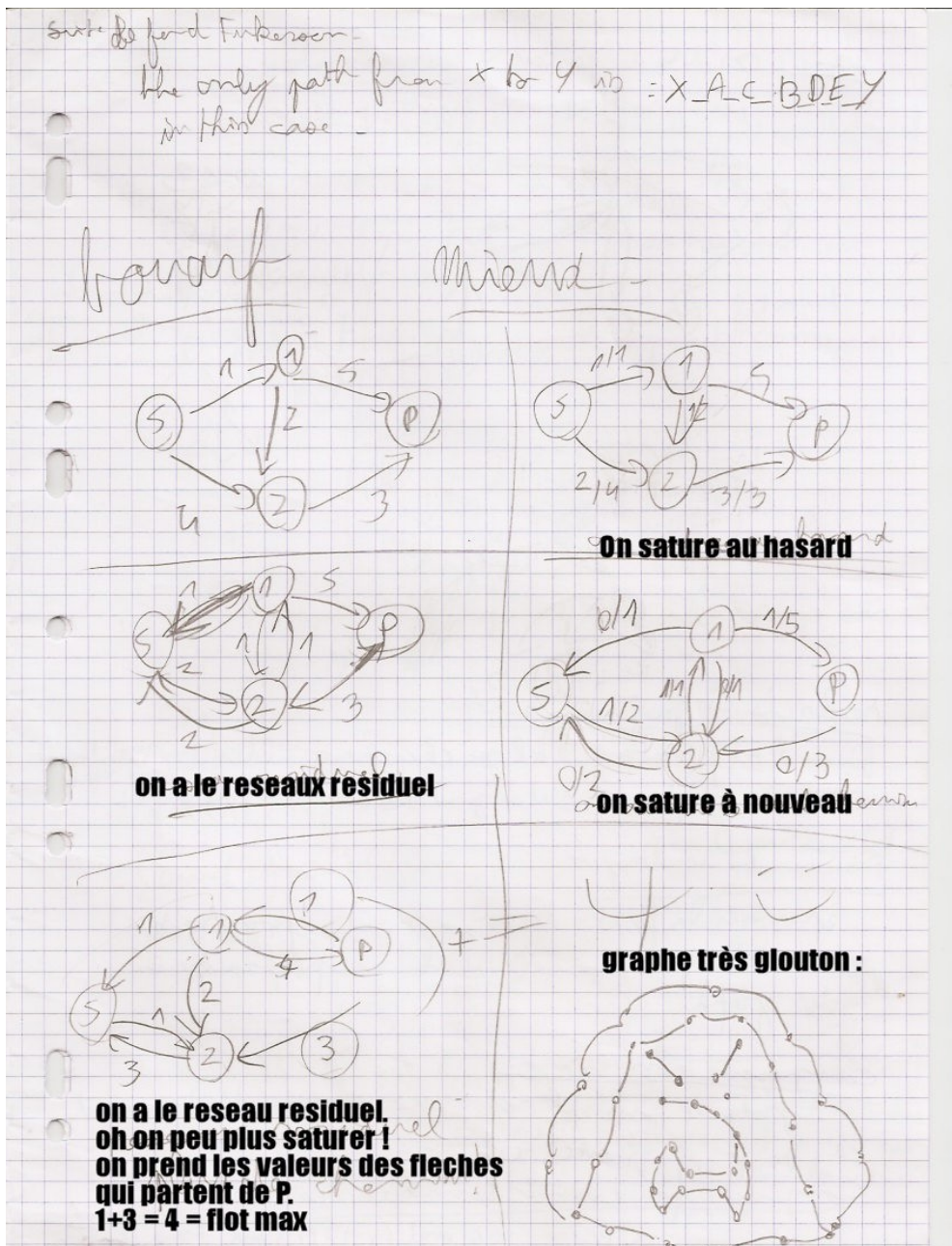
}

Etape 6 : quand y a plus moyen de saturer le reseau residuel, on prend la somme des valeurs des fleches qui partent de la destination, c'est égal au flot max.

« le flot F est max » = « il n'y a pas de chemin augmentant pour F »

(C'était si compliqué que ça à expliquer, V ?)

SCHEMA :



Recherche en profondeur :

ça consiste à faire l'arbre du graphe en allant le plus profond possible. (donc avoir le plus de niveaux possible) ça se fait au jugé.

Recherche en largeur :

pareil mais cette fois on fait tout en parallele, et on ne cherche pas du tout à aller profond. /!\ relier les deux arbres si nécessaire !

Liste d'adjacence :

genre y a 1--->2 et 1---->3 bah on écrit ça : 1->2,3 . Et ainsi de suite pour tous les sommets.

Trouver le flot max :

Consiste à trouver la coupe minimale. C'est à dire la somme des valeurs des arretes TELLES QUE si l'on vire ces

arretes, il n'y a plus aucun lien entre le départ et l'arrivée, ET TELLE QUE cette somme et la plus petite de toutes les sommes que l'on peut trouver.

Select sum(valeur) from arretes where non j'déconne.

Houla c'est fumeux ...

En gros on coupe le graphe en trouvant des arretes qui, si on les vire, font que y a plus moyen depuis le départ d'aller à l'arrivée.

Pour chaque groupe d'arretes trouvées, on fait la somme.

Le but du jeu est de trouver la plus petite somme possible.

Quand on pense l'avoir trouvée, on a win ça donne le flot max.

Algo de Welsh Powell :

probleme : colorier un graphe sans dépasser et foutre de la peinture partout (et accessoirement faire en sorte que deux sommets adjacents n'ont pas la meme couleur)

Etapas :

Determiner le rang des sommets (= nombre de sommets adjacents à ceux-ci)

Pour les sommets qui on le plus haut rang : les colorier au hasard en faisant quand meme gaffe à ce que deux couleurs se touchent pas.

Pour les sommets du rang en dessous : faire pareil

et ainsi de suite jusqu'au plus petit rang.

/!!!!!!\ Toujours utiliser au maximum 4 couleurs !!!

probleme : on veut colorier des aretes

transformer les arretes en sommets

comment ? : on prend un sommet, on prend toutes les arretes qui en partent, et on les retranscrit sur le nouveau graphe comme étant des sommets, et on les relie toutes entre elles par des arretes.

Ça veut dire ? : chaque sommet du nouveau graphe represente une arrete

chaque arrete du nouveau graphe veut dire « relie le meme sommet dans le vieux graphe »

Et apres y a plus qu'a appliquer l'algo de Welsh mon frere Powell.

Le plus long chemin :

appliquer PERT (se demerder)

L'algorithme de Kruskal

est un [algorithme](#) de recherche d'[arbre](#) recouvrant de poids minimum (ARPM) ou [arbre couvrant minimum](#) (ACM).

<=>

L'algorithme de Prim

est un [algorithme](#) déterminant un [arbre couvrant minimal](#) d'un [graphe connexe](#) valué. C'est-à-dire qu'il trouve un sous-ensemble d'arêtes formant un [arbre](#) incluant tous les sommets, tel que la somme des poids de chaque arête soit minimal.