1 Dénombrement et probabilité

1.0.1 Combinaison sans répétition

Parmi n, en choisir r, sans prendre compte de l'ordre et sans répétition :

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

1.0.2 Combinaison avec répétition

Parmi n, en choisir r, sans prendre compte de l'ordre et avec répétition :

$$\binom{n+k-1}{k} = \binom{n+k-1}{n-1}$$

1.0.3 Permutation sans répétition

Parmi n, en choisir r, en prenant compte de l'ordre et sans répétition :

$$\frac{n!}{(n-k)!}$$

1.0.4 Permutation avec répétition

Parmi n, en choisir r, en prenant compte de l'ordre et avec répétition :

$$n^r$$

1.1 Identité de Pascal

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

2 Probabilités

$$E(x) = \sum_{r \in X(s)} p(X = r)r$$

$$E(X+Y) = E(X) + E(Y)$$

$$V(X) = E(X^2) - E(X)^2$$

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

$$p(A|B) = \frac{p(B|A)p(A)}{p(B|A)p(A) + p(B|\bar{A})p(\bar{A})}$$

Pour des variables aléatoires indépendantes X et Y :

$$V(X+Y) = V(X) + V(Y)$$

$$E(XY) = E(X)E(Y)$$

3 Les graphes

Simples graphs : Un graphe sans boucle (sur le même noeud) et sans arête multiple.

Graphes connexes : On parle de graphes connexes lorsque l'on parle de graphe non-orienté.

Complete graphs : Tout les noeuds sont connectés entre eux.

Cycles graphs : Chaque noeud est connecté à deux autre noeuds et ça forme une grande boucle passant par tous les noeuds.

Wheels graphs : Comme un *cycle graph* auquel on ajoute un noeud connecté à tous.

n-Cubes graphs : un graphes qui comporte 2^n noeuds qui sont placé comme un cube (pour le graphe à 8 noeuds.)

Bipartite graphs : Un graphe ou l'on peut séparer sont ensemble de noeuds en deux sous-ensemble V_1 et V_2 de manière à ce qu'il n'y ait que des arêtes entre un noeud de V_1 et un noeud de V_2.

Complete bipartite graphs : C'est un *bipartite graphs* où chaque noeuds est connecté à tous les noeuds de l'autre sous-ensemble.

Graphes isomorphes : Deux graphes isomorphe ont le même nombre de sommets et sont connectés de la même façon.

Connected graphs : un graphe non-directionnel où il existe un chemin entre chacun des noeuds de ce graphe.

Dual graphs : Le graphe *dual* d'un graphe planaire G comporte un noeud pour chaque région de la représentation planaire de G et chacun de ces noeuds est connecté si ce sont des régions adjacentes, plus précisément si ils ont un bord en commun.
Il peuvent aussi avoir plusieurs bords en commun $\rightarrow$ multigraph !
Attention, un graphe planaire peut avoir plusieurs graphes dual *non-isomorphic*.

3.1 Les chemins

Euler path : Un trajet qui parcours toutes les arêtes du graphes exactement une fois.

Euler circuit : Comme un *Euler path* mais qui commence et reviens sur le même noeuds.

Graphes eurélien : On parle de graphe eurélien lorsqu'un graphe contient un *Euler path*.

Un graphe est eulérien si et seulement si chacun de ses sommets a un nombre pair d'arêtes

Graphes hamiltonien : Un graphe hamiltonien est un graphe possédant au moins un cycle passant par tous les sommets une fois et une seule.

Hamilton path : Un chemin qui passe une et une seule fois par tout les sommets.

Hamilton circuit : C'est comme un *Hamilton path* seulement que ce chemin commence et finit sur le même noeud.

3.2 Les graphes planaires

Un graphe planaire est un graphe qui a la particularité de pouvoir se représenter sur un plan sans qu'aucune arête n'en croise une autre.

Un graphe fini est planaire si et seulement s'il ne contient pas de sous-graphe qui est une expansion de K_5 (graphe complet à 5 noeuds) ou $K_{3,3}$ (le graphe complet biparti à 3+3 sommets).

Soit G un graphe planaire de e arêtes et v noeuds. Selon Euler, le nombre de régions sur la représentation planaire de G sera de $r = e - v + 2$.

4 Trees

Les *tree* sont des graphes simple, connecté et sans cycles ou circuit. Dans un arbre, il y a un unique chemin entre chaque pair de noeuds.

Rooted trees : un arbre qui contient un noeud désigné comme la base ou la racine de cette arbre. Toutes les arêtes sont dirigé de manière à s'éloigner de la racine.

Il existe plusieurs moyen de parcourir un arbre en passant par tout les noeuds :

1. **preorder traversal** : On traverse l'arbre depuis sa base en allant toujours vers la droite si possible, sinon on remonte et on continue sur le prochain enfant non visité.
2. **inorder traversal** : On traverse l'arbre depuis la feuille la plus à gauche, on essaye de rester toujours le plus à gauche possible en remontant si nécessaire.
3. **postorder traversal** : On traverse l'arbre depuis la feuille la plus à droite, on ne passe par un parent que lorsque tous ses enfants ont été parcourus.

Spanning tree : Un *spanning tree* d'un graphe G est un sous-graph de G qui est un arbre et qui contient tous les noeuds de ce graphe.

Depth-First Search : C'est un algorithme qui transforme un graphe en *Spanning tree*. Il procède en plusieurs étapes :

1. Choisir un noeud quelconque du graphe qui sera notre base.
2. Ajouter un noeud adjacent au dernier noeud ajouté qui n'a pas encore été choisi.
3. Dès le moment où le dernier noeud ajouté n'a aucun noeud adjacent qui n'ait pas encore été choisi, on remonte dans l'arbre jusqu'à un noeud qui ait des adjacents encore non-choisits.
4. On s'arrête lorsque tous les noeuds du graphe ont été placé dans l'arbre.

Breadth-First Search : C'est un algorithme qui transforme un graphe en *Spanning tree*. IL procède en plusieurs étapes :

1. Choisir un noeud quelconque du graphe qui sera notre base.
2. On ajoute aux derniers noeuds ajoutés (de gauche à droite) tous leurs noeuds adjacents qui n'ont pas encore été choisi.
3. On s'arrête lorsque tous les noeuds du graphe ont été placé dans l'arbre.

Minimum spanning trees : C'est un le *spanning tree* d'un graphe pondéré avec le plus petite somme possible des poids des arêtes.

Prim's algorithm : Un algorithme qui donne le *minum spanning tree* d'un graphe :

1. Prendre l'arête avec le plus petit poids et la mettre dans l'arbre. (Donc on met aussi ses deux sommets)
2. On ajoute à l'arbre l'arête connecté à un noeud avec le plus petit poids possible. (Il ne faut pas que l'autre noeud de cette arête soit déjà dans l'arbre).
3. On s'arrête lorsque tout les noeuds ont été ajoutés.

Kruskal's algorithm : Un algorithme qui donne le *minum spanning tree* d'un graphe :

1. On prend l'arête avec le plus petit poids.
2. On ajoute au fur et à mesure les arêtes qui on le plus petit poids (l'ajout d'une de ces arêtes ne doit pas formé de circuit fermé).
3. On s'arrête quand on a ajouté tous les noeuds du graphe à l'arbre.

5 Complexité

– $\forall a > 0, b > 0, u > v, a, b, u, v$ constants :

$$\log_b(n^v) \text{ est } O(\log_a(n^u))$$

$$\log_a(n^u) \text{ est } O(\log_b(n^v))$$

et ils sont les deux $O(\log(n))$

– $n!$ est $O(n^n)$ mais n^n n'est pas $O(n!)$. Cependant :

$$\log(n!) \text{ est } O(\log(n^n)) = O(n\log(n)) \text{ et } \log(n^n) = n\log(n) \text{ est } O(\log(n!))$$

– $f(x)$ is $o(g(x))$ if $\lim\limits_{x\to\infty} \frac{f(x)}{g(x)} = 0$

6 Fonctions génériques

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

$$\frac{1-x^{n+1}}{1-x} = \sum_{k=0}^{n} x^k$$

$$\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$$

7 Solving Linear Recurrence Relations

On a la relation de récurrence linéaire (de degré 2 par exemple) :

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + f(n)$$

Il en découle la récurrence linéaire homogène associée : On a la relation de récurrence linéaire :

$$a_n = c_1 a_{n-1} + c_2 a_{n-2}$$

Pour résoudre l'équation homogène, on prend l'équation caractéristique de la forme :

$$r^2 - c_1 r - c_2 = 0 \text{ avec les solutions } r_0, r_1$$

On trouve alors une solution de la forme :

$a_n^{(h)} = \alpha_1 r_0^n + \alpha_2 r_1^n$ ou $a_n^{(h)} = \alpha_1 r_0^n + \alpha_2 n r_0^n$ dans le cas où il n'y a qu'une solution à l'eq. carac.

Pour trouver les solutions de l'équation non-homogène, on cherche une solution particulière. On a $f(n) =$ (polynome de degré ts^n). Si s est une solution de l'eq. carac., alors la solution particulière aura la forme :

$$a_n^{(p)} n^m (\text{polynôme de degré t}) s^n \text{ avec } m \text{ la multiplicité de } s$$

Sinon m égal à 0. Ensuite on introduit cette solution particulière dans l'équation de base pour définir les coefficients. Finalement, on additionne $a_n^{(p)}$ et $a_n^{(h)}$ puis on trouve les valeurs des coefficients grâce au conditions initiales.