INF4705: Analyse et conception d'algorithme

Laboratoire 2

Présenté à : Samuel Gagnon

Soumis par

Raphael Christian-Roy(1743344)

Louis-Charles Hamelin(1742949)

Section 01 (B1)

Mardi 28 mars 2017

Introduction

Lors de ce second travail pratique, nous devions implémenter trois versions d'un algorithme calculant le nombre d'extensions linéaires d'un graphe orienté et sans cycle. Le premier algorithme est de type vorace et son implémentation nous ait donné sous forme de pseudo-code. Ce dernier algorithme, une fois implémenté, nous fournit une approximation du nombre d'extensions linéaires possible pour ce graph. Le second algorithme est de type récursif ou l'on fait des retours en arrière pour générer les permutations possibles. Finalement, le dernier algorithme demandé suit le patron de conception de programmation dynamique.

Cadre expérimental

Pour ce qui est de l'environnement de développement et d'analyse, nous avons utilisé un MacBook Pro 2016, ayant 16Gb de mémoire vive et un processeur 3.3 GHz Intel Core i7. La technologie utilisée pour concevoir les algorithmes est P**ython 3.6 avec l'interpréteur Anaconda** afin d'avoir accès à deux librairies fortes utiles pour ce travail, soit networkx et NumPy, qui nous permet de construire des graphes et avoir accès à certains de leurs attributs.

Jeux de données

Dans ce laboratoire, nous avions comme série de données des graphes ou l'on nous donnait les paires de noeuds liés par un arcs. Variant selon la taille du graphe en nombre de noeuds, il y a 230 exemplaires différents. Le nombre de noeuds dans le graphe par fichier est incrémenté de 4 entre 10 et 30 noeuds. Parmi les 40 exemplaires d'une même taille en termes de noeuds, il y a 4 types de séries différentes contenant 10 exemplaires chaque selon la largeur du graphe.

Résultats expérimentaux

Poset10-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	894.427191	1984	y	1984
b	344.265186	716	y	716
c	447.213595	964	y	964
d	1202.813061	784	y	784
e	447.213595	1773	y	1773
f	1202.813061	4190	y	4190
g	601.40653	1954	y	1954
h	231.481481	224	y	224
i	894.427191	560	y	560
j	231.481481	700	y	700

Temps moyen d'exécution de l'algorithme vorace: 0.000138 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 0.015012 seconde

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset10-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	312.5	25200	y	25200
b	304.29031	97920	y	97920
c	98.821177	38160	y	38160
d	312.5	30240	y	30240
e	481.125224	75600	y	75600
f	370.37037	37800	y	37800
g	56.568542	16560	y	16560
h	481.125224	95760	y	95760
i	98.821177	29520	y	29520
j	117.121395	73440	y	73440

Temps moyen d'exécution de l'algorithme vorace: 0.000107 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 0.497701 seconde

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset10-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	60.858062	332640	y	332640
b	60.858062	241920	y	241920
c	79.056942	544320	y	544320
d	250	453600	y	453600
e	250	453600	y	453600
f	250	403200	y	403200
g	79.056942	483840	y	483840
h	250	453600	y	453600
i	19.245009	302400	y	302400
j	60.858062	332640	y	332640

Temps moyen d'exécution de l'algorithme vorace: 0.000084 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 4.249266 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset14-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	30501.59259	62696	y	62696
b	15250.7963	476476	y	476476
c	11177.4113	86868	y	86868
d	11177.4113	42120	y	42120
e	5870.034231	59896	y	59896
f	30501.59259	216482	y	216482
g	1568.832639	298872	y	298872
h	9037.508916	204338	y	204338
i	12867.85938	1313295	y	1313295
j	29463.97009	624812	y	624812

Temps moyen d'exécution de l'algorithme vorace: 0.00019 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 4.045681 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset14-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	18665.66456	4406248	y	4406248
b	62996.6179	1595664	y	1595664
c	27512.64062	13588848	y	13588848
d	27512.64062	14414400	y	14414400
e	48494.81907	7316400	y	7316400
f	50202.64444	55648320	y	55648320
g	50202.64444	50761620	y	50761620
h	84716.96249	46546500	y	46546500
i	32607.57407	27220788	y	27220788
j	84716.96249	7657944	y	7657944

Temps moyen d'exécution de l'algorithme vorace: 0.00021 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 249.051659 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset14-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	181132.4849	x	y	390810420
b	69717.92593	x	y	224071848
c	3234.508658	x	y	126486360
d	31443.01785	x	y	85044960
e	8999.516842	x	y	52972920
f	117649	x	y	14414400
g	69717.92593	x	y	226017792
h	37265.79893	x	y	183338064
i	18632.89947	x	y	320938800
j	16807	x	y	160095936

Temps moyen d'exécution de l'algorithme vorace: 0.000197 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long.

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extension linéaire n'a pas pu être calculé avec l'algorithmes de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset18-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	247949.113	427998	y	427998
b	244379.2977	16370420	y	16370420
c	97374.17183	17471078	y	17471078
d	1955034.382	3944424	y	3944424
e	433149.2746	2571720	y	2571720
f	752493.0843	3663532	y	3663532
g	121215.8437	2842662	y	2842662
h	291246.5151	4705201	y	4705201
i	1580745.325	17149960	y	17149960
j	189170.1606	726430	y	726430

Temps moyen d'exécution de l'algorithme vorace: 0.000296 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 90.660296 seconde

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset18-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	1580745.325	x	y	229764012
b	3587226.75	x	y	312153020
c	2761448.44	x	y	430848372
d	10670030.95	x	y	3348322302
e	1301759.278	x	y	703743660
f	8503056	x	y	164685326
g	7174453.5	x	y	251090685
h	1793613.375	x	y	35678604
i	12645962.6	x	y	1068066576
j	816754.105	x	y	1521364158

Temps moyen d'exécution de l'algorithme vorace: 0.000341 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long.

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extension linéaire n'a pas pu être calculé avec l'algorithmes de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset18-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	558877.8695	x	y	24747615360
b	57395628	x	y	2.34984E+11
c	6377292	x	y	78110343360
d	797161.5	x	y	8633999304
e	11414485.62	x	y	1.02115E+11
f	5029900.825	x	y	16228051812
g	3650574.957	x	y	63006862374
h	6534032.84	x	y	18730568448
i	193710244.5	x	y	22096649664
j	74559107.87	x	y	56174898528

Temps moyen d'exécution de l'algorithme vorace: 0.000305 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long.

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extension linéaire n'a pas pu être calculé avec l'algorithmes de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset22-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	5206116.182	4354368	y	4354368
b	35141284.23	61053356	y	61053356
c	52631754.46	202860	y	202860
d	68718623.21	105288250	y	10528825
e	20638865.06	70768494	y	70768494
f	22824933.65	40080441	y	40080441
g	35141284.23	9261420	y	9261420
h	452979.2057	24393680	y	24393680
i	1394814.221	28459166	y	28459166
j	602721.363	5487543	y	5487543

Temps moyen d'exécution de l'algorithme vorace: 0.000305 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 549.891235 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset22-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	426596781	x	y	3.73015E+11
b	189030011.9	x	y	1.05377E+11
c	756120047.7	x	y	1.7825E+11
d	245557188.6	x	y	6686206418
e	554165474.2	x	y	36713234838
f	359941033.9	x	y	30195038032
g	179970517	x	y	1.61552E+11
h	179970517	x	y	59212119966
i	372616953.6	x	y	28340662596
j	245557188.6	x	y	18967237166

Temps moyen d'exécution de l'algorithme vorace: 0.000459 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset22-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	992767403	x	y	3.82712E+12
b	1164122968	x	y	2.57245E+12
c	8134556535	x	y	3.72107E+12
d	714144881	x	y	21491823780
e	8996168763	x	y	1.73872E+13
f	26457340.19	x	y	8.71553E+11
g	25189339.74	x	y	1.53815E+12
h	1792284558	x	y	2.78537E+12
i	5283549456	x	y	3.55849E+12
j	1257582218	x	y	5.20244E+12

Temps moyen d'exécution de l'algorithme vorace: 0.000437 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset26-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	61567107.51	5027586016	y	5027586016
b	811458083.1	192937716	y	192937716
c	176214249.8	1623284724	y	1623284724
d	88107124.88	383117122	y	383117122
e	210524317.2	7536060	y	7536060
f	217938279.7	45968140	y	45968140
g	672032112.2	4727173643	y	4727173643
h	660036545.3	157788704	y	157788704
i	1275166008	246024216	y	246024216
j	489086159	17625766189	y	17625766189

Temps moyen d'exécution de l'algorithme vorace: 0.000365 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 704.36004 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset26-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	5970290029	x	y	1.46803E+12
b	54245704174	x	y	8.51611E+12
c	835386072.9	x	y	2.43622E+12
d	15387649575	x	y	4.79896E+11
e	1381649575	x	y	2.00579E+11
f	1670772146	x	y	5.44106E+12
g	1.08491E+11	x	y	6.18711E+11
h	1298340996	x	y	1.11144E+11
i	1430306653	x	y	13345674180
j	25083308409	x	y	1.74036E+11

Temps moyen d'exécution de l'algorithme vorace: 0.000374 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset26-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	2.00666E+11	x	y	1.56014E+13
b	70467236791	x	y	2.19591E+12
c	9118607156	x	y	5.07966E+12
d	74609077044	x	y	3.79248E+12
e	6683088584	x	y	1.90372E+12
f	1.51621E+12	x	y	4.68255E+13
g	56156056491	x	y	5.29772E+11
h	50166616817	x	y	7.27328E+12
i	24230008527	x	y	8.40855E+11
j	21447846901	x	y	3.8701E+13

Temps moyen d'exécution de l'algorithme vorace: 0.000437 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset30-4

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	80368727203	41605488	y	41605488
b	2332800000	12012774390	y	12012774390
c	6006774902	22840199949	y	22840199949
d	276681548.5	1938339394	y	1938339394
e	4829906831	12557286180	y	12557286180
f	10046090900	507504486	y	507504486
g	1734699946	233076312	y	233076312
h	2299213814	1281428588	y	1281428588
i	7733484381	497333232	y	497333232
j	5636555835	20396670388	y	20396670388

Temps moyen d'exécution de l'algorithme vorace: 0.000425 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: 834.00546 secondes

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Poset30-6

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	5.98361E+12	x	y	1.34912E+13
b	1.32382E+12	x	y	3.49677E+13
c	2.04951E+11	x	y	5.02548E+12
d	2.54769E+11	x	y	5.72399E+11
e	1.15226E+11	x	y	9.02273E+11
f	1.4665E+11	x	y	1.58277E+12
g	3.10918E+13	x	y	4.49086E+12
h	2.14075E+12	x	y	1.38738E+12
i	7.68867E+11	x	y	3.40385E+13
j	5.80395E+12	x	y	1.35662E+15

Temps moyen d'exécution de l'algorithme vorace: 0.000519 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Poset30-8

	Vorace	Retour arrière	Programmation dynamique	Théorique
	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire	Nombre d'extension linéaire
a	2.69673E+13	x	y	4.7447E+16
b	5.94169E+12	x	y	9.10184E+14
c	3.9214E+13	x	y	3.66221E+16
d	1.94324E+12	x	y	3.55802E+15
e	1.11237E+13	x	y	3.97757E+14
f	2.89003E+12	x	y	7.04642E+14
g	1.23019E+13	x	y	1.22008E+15
h	6.25E+13	x	y	1.16102E+15
i	1.59806E+13	x	y	3.70727E+15
j	5.80948E+12	x	y	4.42627E+15

Temps moyen d'exécution de l'algorithme vorace: 0.000549 seconde

Temps moyen d'exécution de l'algorithme de retour en arrière: Trop long

Temps moyen d'exécution de l'algorithme de programmation dynamique: Y secondes

Le nombre d'extensions linéaire n'a pas pu être calculé avec l'algorithme de retour en arrière pour cette série, car ce dernier prend beaucoup trop de temps.

Temps moyen des algorithmes en seconde pour 10 exemplaires

	Vorace	Retour arrière	Programmation dynamique
Poset10-4	0.000138	0.015012	y
Poset10-6	0.000107	0.497701	y
Poset10-8	8.40E-05	4.249266	y
Poset14-4	0.00019	4.249266	y
Poset14-6	0.00019	249.051659	y
Poset14-8	0.000197	Trop long	y
Poset14-10	0.000227	Trop long	y
Poset18-4	0.000296	90.660296	y
Poset18-6	0.000341	Trop long	y
Poset18-8	0.000305	Trop long	y
Poset18-10	0.000358	Trop long	y
Poset22-4	0.000305	549.891235	y
Poset22-6	0.000459	Trop long	y
Poset22-8	0.000437	Trop long	y
Poset22-10	0.000395	Trop long	y
Poset26-4	0.000365	704.36004	y
Poset26-6	0.000374	Trop long	y
Poset26-8	0.000437	Trop long	y
Poset26-10	0.000441	Trop long	y
Poset30-4	0.000425	834.00546	y
Poset30-6	0.000519	Trop long	y
Poset30-8	0.000549	Trop long	y
Poset30-10	0.000525	Trop long	y

Analyse et discussion

I) Tentez une analyse asymptotique du temps de calcul pour chaque algorithme.

Algorithme vorace

Meilleur cas: $\Omega(V + A)$
Moyen cas: $\Theta(V + A)$
Pire cas: $O(V + A)$

Car on a deux boucles for qui sont une à la suite de l'autre et non imbriquée dans l'algorithme.

Algorithme de retour en arrière

Meilleur cas: $\Omega(n)$
Moyen cas: $\Theta(nlog(n))$
Pire cas: $O(n^2)$

Car on a deux boucles for imbriquées qui sont une à la suite de l'autre de dans l'algorithme.

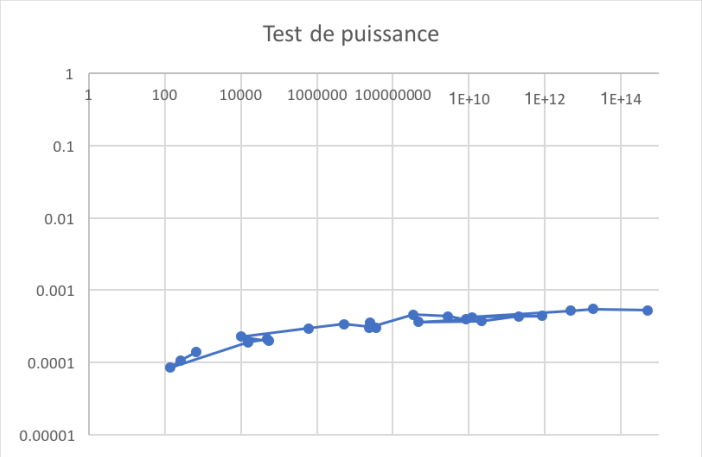
Algorithme de programmation dynamique

Meilleur cas: $\Omega(n^c)$
Moyen cas: $\Theta(n^c)$
Pire cas: $O(n^c)$
Ou C est le nombre de chaînes trouvées.

II) Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix (cette méthode peut varier d'un algorithme à l'autre). Justifiez ces choix.

Algorithme vorace

Nous allons utiliser le test de puissance, car nous ne connaissons pratiquement rien de la consommation des ressources de notre algorithme. Ce test nous aidera à exprimer la consommation en fonction du nombre d'extensions linéaire.



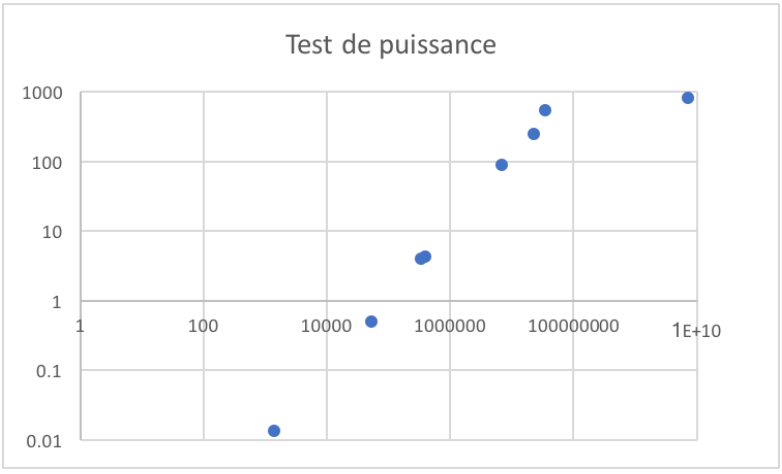
Le graphique ci-haut nous illustre le test de puissance pour l'algorithme vorace. Ce dernier tend vers une valeur constante, ce qui signifie que notre algorithme croît de manière sub-linéaire. Ainsi, on peut donc conclure que notre hypothèse semble valide, car nous avions déterminé que cet algorithme a une notation asymptotique dans l'ordre de V+A.

L'image ci-dessous démontre notre hypothèse:

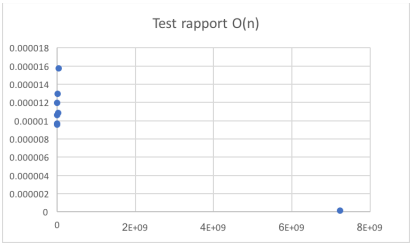
```
1  forall the v ∈ V do
2  |  pred[v] ← −1;
3  initialize queue Q with {v ∈ V : indegree(v) = 0};
4  last ← −1;
5  while Q is not empty do
6  |  remove vertex u from the front of Q;
7  |  last ← u;
8  |  forall the arcs (u, v) ∈ A do
9  |  |  pred[v] ← u;
10 |  |  if Q contains v then
11 |  |  |  move v to the end of Q;
12 |  |  else
13 |  |  |  add v at the end of Q;
14 c ← ⟨⟩;
```

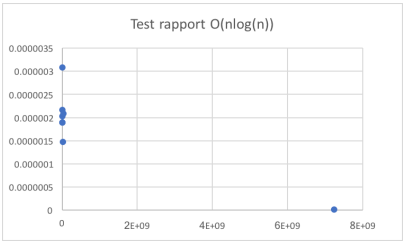
Algorithme de retour en arrière

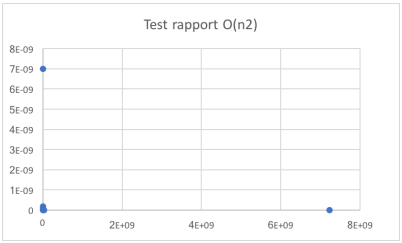
Nous utiliserons le test de puissance comme pour l'algorithme vorace et le test du rapport afin préciser l'analyse asymptotique de notre algorithme de retour en arrière, soit une fonction récursive. On a une certaine idée de la consommation de cet algorithme, donc nous placerons les couples (x, y/f(x)) sur un graphique d'échelle normale.



Le test de puissance ci-haut nous indique que la consommation de notre algorithme croît de façon polynomiale, en raison de la tendance linéaire de nos données.







Si une courbe passant par ces points converge vers b > 0, l'hypothèse de f(x) est sensée et b représente une constante multiplicative. Dans le cas de n, lorsqu'on zoom sur la valeur près de l'axe des x, on peut remarquer que la valeur est légèrement supérieure à 0. La raison pourquoi nous sommes si près de 0 est que notre unité de mesure est en seconde et que nos temps sont rapides tandis que le nombre d'extensions linéaires est très élevé.

Si b converge vers 0, alors notre hypothèse est une surestimation de la consommation de l'algorithme. Alors, dans le cas de n^2 et nlg(n), on peut remarquer une convergence en 0.

III) Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

Tout d'abord, on peut remarquer que l'approximation par l'algorithme vorace est très rapide en tout temps, par contre sa précision laisse à désirer. En effet, cette dernière est peu précise pour le nombre d'extensions linéaires approximées comparativement aux résultats théoriques qui nous sont fournis.

Pour ce qui est de l'algorithme de retour en arrière, il est très précis en comparant le nombre d'extensions linéaires calculées avec la théorie. Par contre, ce dernier consomme beaucoup et a un temps de calcul très élevé lorsque le nombre de noeuds est supérieur à 10 et une largeur de graphe supérieure à 4.

Quant à l'algorithme de programmation dynamique, ce dernier était très difficile à implémenter selon nous. En effet, la création de cet algo devait se séparer en trois parties. La première consistait à prendre le graph généré à partir des fichiers de données et d'y appliquer l'algorithme de fermeture transitive. Ensuite, nous devions prendre ce graphe et lui appliquer l'algo précédent afin de trouver les plus longues chaînes. Tout dépendant le nombre de chaînes trouvées, notre algo pouvais prendre des formes différentes. Étant formé d'une boucle for et d'un appel récursif, le nombre de chaînes trouvées détermine le nombre de fois que nous appliquerons cette boucle for afin de remplir un tableau qui a autant de dimensions que de chaînes trouvées. Nous pouvons finalement remplir le tableau dynamique et ainsi trouver le nombre de combinaisons à la fin du tableau.

N.B. Comme vous pourrez remarquer, même après 20h passé dessus, nous n'avons pas réussi à finir l'algorithme #3 et à le faire fonctionner. Or, afin de perdre le moins de points possible, nous avons fait le rapport et tout le reste du labo toujours en essayant de mettre le plus de détails, même si nous n'avons pas pu le finir.

IV) Indiquez sous quelles conditions vous utiliseriez l'un de ces algorithmes plutôt que les deux autres.

L'algorithme vorace serait utile dans le cas ou nous devons analyser un graphe ayant plus de 14 noeuds et une largeur de graphe supérieur à 4 dans le but de seulement avoir une approximation afin d'éviter un temps de calcul relativement élevé. Même si cette dernière n'est pas juste, elle peut donner un ordre de grandeur du nombre d'extensions linéaires du graphe en question.

D'un autre côté, l'algorithme de retour en arrière serait fort utile dans le cas ou nous voulons un résultat précis du nombre d'extensions linéaires d'un graphe et que nous avons beaucoup de temps pour le calculer. On remarque que le temps d'exécution de cet algorithme est très élevé pour des graphes ayant plus de 14 noeuds et une largeur supérieure à 6.