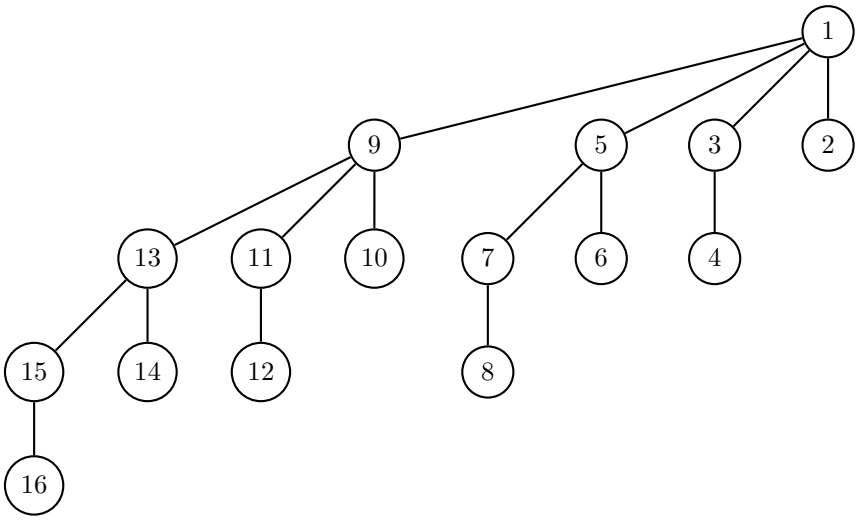
Corrigé de l'épreuve d'informatique 2011
X - ENS MP option informatique

La phrase "tous les nœuds de $\mathcal{T}$ sont supposés distincts" est curieuse, puisque les éléments d'un ensemble sont toujours deux à deux distincts ! On faut donc plutôt comprendre que les valeurs stockées en les différents nœuds de $\mathcal{T}$ sont deux à deux distinctes, ce qui permet sans ambiguïté d'identifier un nœud avec la valeur qu'il contient. Ainsi, le i-ème fils de $\mathcal{T}$ est l'arbre $\mathcal{T}_i$, que l'on peut identifier à la valeur r_i.

Nous noterons $p(s, \mathcal{T})$ la profondeur d'un nœud s dans $\mathcal{T}$, $p(\mathcal{T})$ la profondeur de $\mathcal{T}$, $D(\mathcal{T})$ son diamètre (défini à la question 14), i.e. la longueur maximale d'un chemin de $\mathcal{T}$.

Partie I. Arbres binomiaux

Q1 On obtient directement :



Q2 Notons n_k (resp. e_k) le nombre de nœuds (resp. de nœuds externes) de $\mathcal{B}_k$. Les suites (n_k) et (e_k) vérifient les récurrences :

$$\begin{cases} n_0 = 1 \text{ et } e_0 = 1 \\ \forall k \in \mathbb{N},\ n_{k+1} = 1 + n_0 + n_1 + \cdots + n_k \\ \forall k \in \mathbb{N},\ e_{k+1} = e_0 + e_1 + \cdots + e_k \end{cases}$$

Nous en déduisons

$$\begin{cases} n_0 = e_0 = 1 \\ n_1 = 2 \text{ et } e_1 = 1 \\ \forall k \geq 1,\ n_{k+1} = 2n_k \text{ et } e_{k+1} = 2e_k \end{cases}$$

d'où

$$\begin{cases} \forall k \in \mathbb{N},\ n_k = 2^k \\ e_0 = 1 \\ \forall k \geq 1,\ e_k = 2^{k-1} \end{cases}$$

Q3 Si $\mathcal{B}'_{k-1}$ est une copie de $\mathcal{B}_k$ (les étiquettes de $\mathcal{B}'_{k-1}$ et de $\mathcal{B}_{k-1}$ sont choisies deux à deux distinctes), on obtient $\mathcal{B}_k$ en ajoutant $\mathcal{B}'_{k-1}$ comme premier fils de $\mathcal{B}_{k-1}$, puisque cela revient à choisir $\mathcal{T}_{k-1} = \mathcal{B}'_{k-1}$ dans la construction de $\mathcal{B}_k$.

Q4 La fonction `copie` utilise la fonction auxiliaire récursive `copie_liste` qui, appliquée à une liste d'arbres $[\mathcal{T}_1; \ldots; \mathcal{T}_n]$, renvoie la liste $[\mathcal{T}'_1; \ldots; \mathcal{T}'_n]$ où $\mathcal{T}'_j$ est une copie de $\mathcal{T}_j$ dans laquelle chaque nœud de numéro i est remplacé par un nœud de numéro $i + n$.

```
let copie n t =
   let rec copie_liste = function
       [] -> []
      |Noeud(i,l)::q -> Noeud(i+n,copie_liste l)::(copie_liste q) in
    hd (copie_liste [t]);;
```

Q5 Nous numérotons les nœuds de $\mathcal{B}_k$ de 1 à 2^k (comme à la question 8). La construction est récursive : la copie de $\mathcal{B}_{k-1}$ est obtenue en ajoutant 2^{k-1} à toutes les clés de $\mathcal{B}_{k-1}$. La procédure `bin_aux` renvoie le couple $(2^k, \mathcal{B}_k)$ et évite le calcul de 2^{k-1}.

```
let bin k =
   let rec bin_aux = function
      0 -> 1,Noeud(1,[])
      |k -> let a,t = bin_aux (k-1) in
          match t with
             Noeud(b,l) -> 2*a,Noeud(b,(copie a t)::l) in
   snd (bin_aux k);;
```

Q6 $p(\mathcal{B}_0) = 0$ pour $k \in \mathbb{N}$, $p(\mathcal{B}_{k+1}) = 1 + p(\mathcal{B}_k)$ (d'après la question 3), donc $p(\mathcal{B}_k) = k$ pour tout $k \geq 0$.

Soit $k \geq 2$ et $\mathcal{B}_k = (r_k, (\mathcal{T}_{k-1}, \ldots \mathcal{T}_0))$. En choisissant des nœuds s_0 et t_0 dans $\mathcal{T}_{k-1}$ et $\mathcal{T}_{k-2}$ de profondeur $k-1$ et $k-2$, `chemin`(s_0, t_0) (dans $\mathcal{B}_k$) est de longueur $(k-1)+1+1+(k-2)$, soit $2k-1$.

Si maintenant s et t sont deux nœuds de $\mathcal{B}_k$, le chemin $(x_0, x_1, \ldots, x_l)$ qui les relie passe par un unique nœud x_i de profondeur minimale. Trois cas sont possibles :

- si $i = 0$, $l = p(x_l, \mathcal{B}_k) - p(x_0, \mathcal{B}_k) \leq p(x_l, \mathcal{B}_k) \leq p(\mathcal{B}_k) = k \leq 2k - 1$;
- si $i = l$, $l = p(x_0, \mathcal{B}_k) - p(x_l, \mathcal{B}_k) \leq p(x_0, \mathcal{B}_k) \leq p(\mathcal{B}_k) = k \leq 2k - 1$;
- sinon, les arbres enracinés en x_{i-1} et en x_{i+1} sont des arbres binomiaux $\mathcal{T}_{k_1}$ et $\mathcal{T}_{k_2}$ d'ordres k_1, k_2 distincts et strictement inférieurs à k. Nous avons alors

$$l = p(x_0, \mathcal{T}_{k_1}) + 1 + 1 + p(x_l, \mathcal{T}_{k_2}) \leq k_1 + 2 + k_2 < (k-1) + 2 + (k-1) = 2k$$

Comme $D(\mathcal{B}_1) = 1$, nous avons $D(\mathcal{B}_0) = 0$ et $D(\mathcal{B}_k) = 2k - 1$ pour tout $k \geq 1$.

Q7 Pour $k, \ell \in \mathbb{N}$, notons $n_{k,\ell}$ le nombre de nœuds de $\mathcal{B}_k$ de profondeur ℓ. Nous avons $n_{k,\ell} = 0$ si $\ell > k$, $n_{0,\ell} = 1$ et la question 3 donne, pour $k \geq 1$:

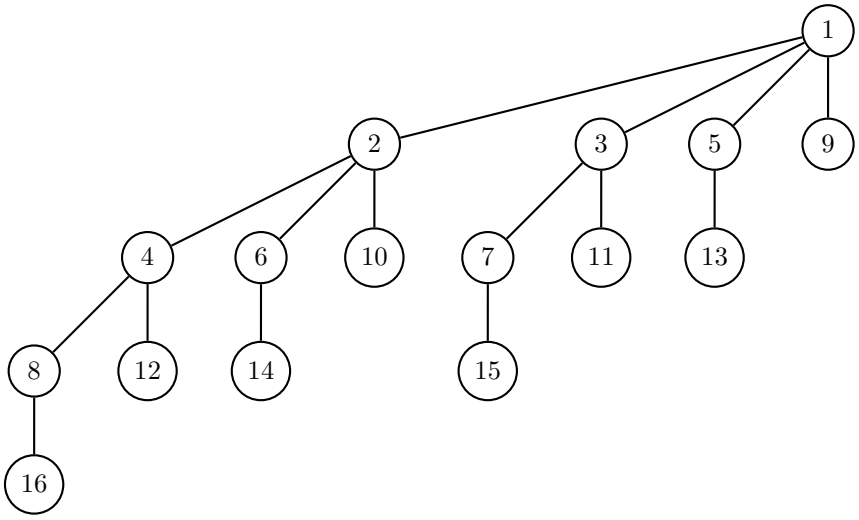
$$\begin{cases} n_{k,0} = 1 \\ \forall l \in \{1, \ldots, k\},\ n_{k,\ell} = n_{k-1,\ell-1} + n_{k-1,\ell} \end{cases}$$

On reconnaît les relations caractérisant les nombres binomiaux, soit :

$$\forall k, \ell \in \mathbb{N},\ n_{k,\ell} = \begin{cases} \binom{k}{\ell} & \text{si } 0 \leq \ell \leq k \\ 0 & \text{sinon} \end{cases}$$

Q8 Il suffit de calculer le tableau des pères des nœuds de $\mathcal{C}_n$ puis de construire l'arbre :

i	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\lceil log_2\, i \rceil$	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4
$i - 2^{\lceil log_2\, i \rceil}$	1	1	2	1	2	3	4	1	2	3	4	5	6	7	8



La définition de $\mathcal{C}_n$ conduit bien à un arbre car tout nœud i autre que 1 a un père compris entre 1 et $i-1$. Le graphe $\mathcal{C}_n$ est donc un arbre de racine 1.

Comme $\mathcal{C}_{16}$ est isomorphe à l'arbre $\mathcal{B}_4$ construit à la question 1, il semble raisonnable de conjecturer que $\mathcal{C}_{2^k}$ est un arbre binomial d'ordre k (la numérotation des sommets ne correspond pas à celle obtenue à la question 5, mais $\mathcal{B}_k$ n'est défini que par sa structure topologique, i.e. pas sa "forme").

Q9 Pour calculer la table des pères sans calculer chaque $\lceil log_2\, i \rceil$, et ainsi assurer le temps de calcul en temps $O(n)$, nous utilisons la méthode suivante. La ligne des $\lceil log_2\, i \rceil$ du tableau précédent est de la forme

$$L = (1, 2, 2, 4, 4, 4, 4, \ldots, \underbrace{2^\ell, \ldots, 2^\ell}_{\ell\ termes}, \ldots).$$

Nous créons donc un tableau P de longueur $n+1$ [1] que nous remplissons, à partir de la case $P.(2)$, par cette suite de valeurs : a est une référence qui contient la puissance de 2 à insérer et b est une référence qui contient le nombre de fois que cette valeur doit encore être insérée. Quand b vaut 1, on multiplie a par 2 et on réinitialise b à la (nouvelle) valeur de a. Sinon, on décrémente b. Une fois le tableau des pères construits, on utilise la méthode suivante :

1. Comme l'indice varie de 2 à n, nous n'utiliserons pas la première case de P.

- on crée un vecteur T de taille $n+1$: pour $1 \leq i \leq n$, $T.(i)$ contiendra, à la fin du calcul, la liste des fils du nœud i dans l'arbre $\mathcal{C}_n$: au début du calcul, chaque case de T contient la liste vide ;

- pour i décroissant de n à 2, on note j le père de i et on ajoute à la liste $T.(j)$ l'arbre `Noeud`$(i, T.(i))$. En effet, les clés des nœuds décroissant le long des branches de $\mathcal{C}_n$, la case $T.(i)$ contient bien la liste des fils de i dans $\mathcal{C}_n$ au moment où on étudie le i-ème nœud (invariant de boucle : au début de la boucle, chaque case $T.(\ell)$, avec $1 \leq \ell \leq n$, contient la liste des fils de ℓ dans $\mathcal{C}_n$ dont la racine est strictement plus grande que i) ;

- à la fin de la boucle, il suffit de renvoyer `Noeud`$(1, T.(1))$.

Cela donne :

```
let c n = let P = make_vect (n+1) 0 and T = make_vect (n+1) [] in
   let a=ref 1 and b=ref 1 in
      for i=2 to n do
         P.(i)<-i-(!a);
         if (!b)=1 then
             begin
             a := 2*(!a);
             b := (!a)
             end
          else
             b := (!b)-1
      done;
   for i=n downto 2 do
      let j=P.(i) in
         T.(j)<-Noeud(i,T.(i))::(T.(j))
   done;
   Noeud(1,T.(1));;
```

Les deux boucles successives sont de longueur $n-1$ et le temps de calcul est constant pour chaque passage, ce qui donne une complexité $O(n)$.

Partie II. Fonctions élémentaires sur les arbres

Q10 La fonction auxiliaire `prof_liste` calcule le maximum des profondeurs des arbres d'une liste et la fonction principale se contente d'appliquer la fonction auxiliaire à la liste constituée par l'unique arbre passé en argument :

```
1 let profondeur t =
2    let rec prof_liste = function
3       [] -> -1
4      |Noeud(_,fils)::freres -> max (prof_liste freres) (1+prof_liste fils) in
5   prof_liste [t];;
```

Cet algorithme effectue deux types de calculs :

- pour chaque nœud, il faut un temps constant pour effectuer le calcul de la ligne 4

- le passage par la ligne 3 est fait une fois pour chaque feuille (quand la liste des fils est vide) et une fois pour chaque nœud qui ne possède pas de "frère droit" (car la liste des frères est vide)

Le temps de calcul est donc un $O(nbn(\mathcal{T}))$.

Q11 On reprend la même analyse, mais la procédure auxiliaire renvoie maintenant un couple (α, p) où α est le numéro d'un nœud (nécessairement externe) de profondeur maximale et p cette profondeur maximale.

```
let noeud_externe_max t =
  let rec max_liste = function
      [] -> 0,(-1)
     |Noeud(a,fils)::freres ->
        let a1,p1 = (max_liste fils) and a2,p2 = (max_liste freres) in
           match p1,p2 with
               (-1),(-1) -> a,0
              |(-1),_ -> a2,p2
              |_ -> if p1+1>p2 then a1,(p1+1) else a2,p2 in
  fst (max_liste [t]);;
```

Q12 Un nouvelle fois la fonction auxiliaire `chemin_liste`, appliquée à une liste d'arbre L, renvoie l'unique chemin, s'il existe, qui relie la racine d'un des arbres de L au sommet s. Si un tel chemin n'existe pas, la fonction renvoie la liste vide.

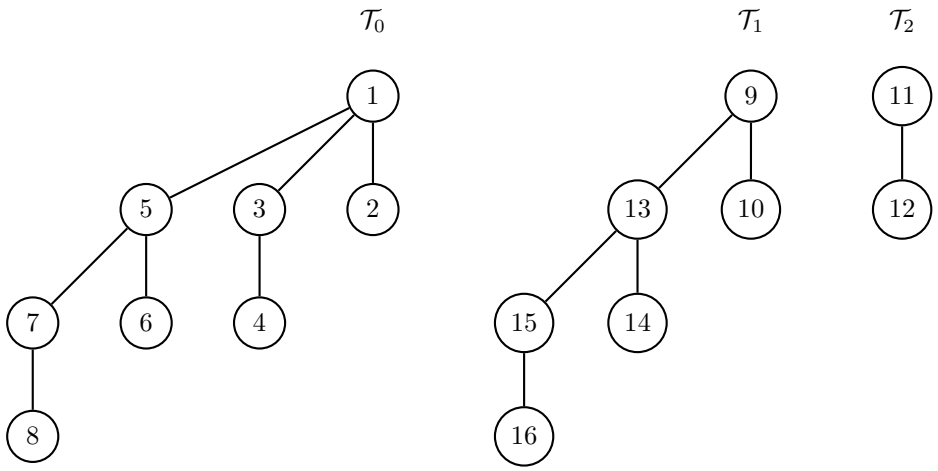
```
let chemin t s =
   let rec chemin_liste = function
       [] -> []
      |Noeud(a,fils)::freres -> if a=s then [s] else
                                begin
                                match chemin_liste fils with
                                   [] -> chemin_liste freres
                                   |c -> a::c
                                end in

      chemin_liste [t];;
```

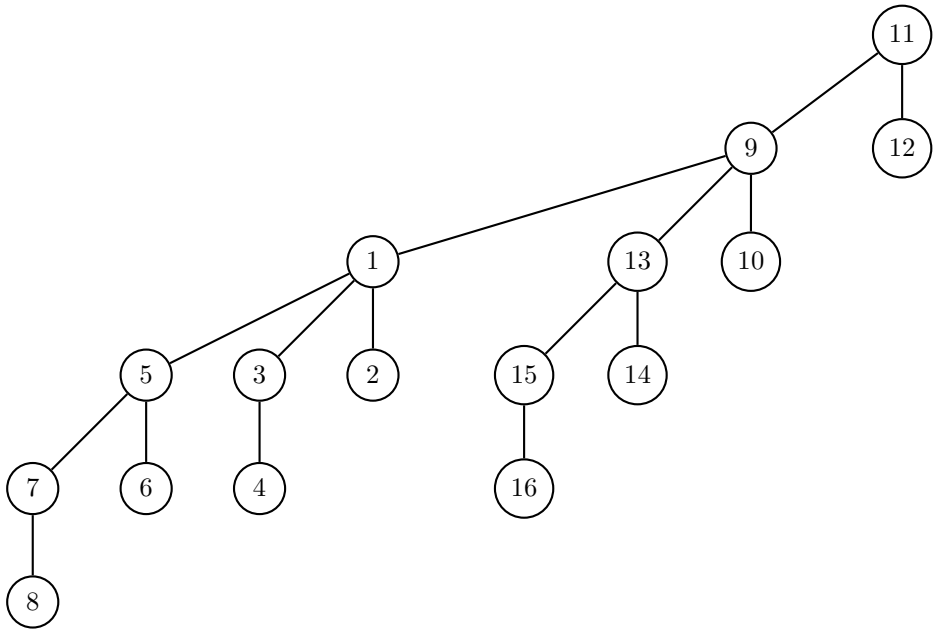
Q13 Je n'ai pas trouvé de méthode simple effectuant le travail en parcourant une seule fois l'arbre, mais l'énoncé introduisant la fonction `chemin`, c'est peut-être la méthode suivante qui est attendue par le concepteur du sujet. On effectue deux parcours de l'arbre à la recherche de la valeur s et le temps de calcul est bien en $O(nbn(\mathcal{T}))$.

1. On commence par calculer le chemin $[x_0, x_1, \ldots, x_k]$ permettant d'aller de r à s ;
2. Une fois ce chemin connu, on calcule la liste d'arbres $(\mathcal{T}_0, \ldots, \mathcal{T}_{k-1}, \mathcal{T}_k)$ où :
   - pour i compris entre 0 et $k-1$, $\mathcal{T}_i$ est le sous-arbre de $\mathcal{T}$ enraciné en x_i dont on a supprimé le sous-arbre enraciné en x_{i+1}.
   - $\mathcal{T}_k$ est le sous-arbre de $\mathcal{T}$ enraciné en s ;

   Ainsi, pour l'arbre $\mathcal{B}_4$ dessiné à la question 1 et $s = 11$, on aura $x_0 = 1$, $x_1 = 9$, $x_2 = 11$ et par exemple :



3. Il reste alors à construire l'arbre $pivot(s, \mathcal{T})$: on ajoute $\mathcal{T}_0$ comme fils à $\mathcal{T}_1$, puis le nouvel arbre $\mathcal{T}_1$ comme fils à $\mathcal{T}_2$, et ainsi de suite jusqu'à $\mathcal{T}_k$. Dans l'exemple précédent, nous obtenons :



Pour construire la liste $(\mathcal{T}_i)$, nous utilisons quelques fonctions auxiliaires :

- si $\mathcal{T}_1$ et $\mathcal{T}_2$ sont deux arbres, l'appel `ajouter_fils` $\mathcal{T}_1$ $\mathcal{T}_2$ renvoie l'arbre obtenu en ajoutant $\mathcal{T}_1$ comme fils (le plus à droite) à l'arbre $\mathcal{T}_2$;
- si $\mathcal{T}$ est un arbre et si a est la valeur d'un des fils de la racine de $\mathcal{T}$, `supprimer_fils` $\mathcal{T}$ a renvoie le couple $(\mathcal{T}_1, \mathcal{T}_2)$ où $\mathcal{T}_1$ est le fils de $\mathcal{T}$ dont la racine est égale à a et $\mathcal{T}$ est l'arbre obtenu en supprimant $\mathcal{T}_1$ de $\mathcal{T}$;
- si $\mathcal{T}$ est un arbre, s un nœud de $\mathcal{T}$ et $[x_0, \ldots, x_k]$ un chemin de r à s, l'appel `calcul_liste` $\mathcal{T}$ $[x_1; \ldots; x_k]$ renvoie la liste $[\mathcal{T}_0; \mathcal{T}_1; \ldots; \mathcal{T}_k]$;

- la fonction `regrouper` fait le travail de la troisième étape.

```
let ajouter_fils t = function
   Noeud(a,fils) -> Noeud(a,t::fils);;

let rec supprimer_fils t a = match t with
   Noeud(_,[]) -> failwith "a n'est pas un fils"
   |Noeud(b,Noeud(c,fils)::q) -> if c=a then
                                    Noeud(c,fils),Noeud(b,q)
                                 else
                                    begin
                                    let t1,t2 = supprimer_fils (Noeud(b,q)) a in
                                       t1,ajouter_fils (Noeud(c,fils)) t2
                                    end ;;

let rec calcul_liste t = function
   [] -> [t]             (* la racine de t est la valeur pivot s *)
   |a::q -> let t1,t2 = supprimer_fils t a in
      t2::(calcul_liste t1 q);;

let rec regrouper = function
   [t] -> t
   |t1::t2::q -> regrouper ((ajouter_fils t1 t2)::q);;
```

Il ne reste qu'à écrire la fonction principale :

```
let pivot t s = regrouper (calcul_liste t (tl(chemin t s)));;
```

Q14 Pour s et t nœuds de $\mathcal{T}$, nous noterons $d(s,t)$ la longueur de `chemin`(s,t). Les chemins étant les mêmes dans $\mathcal{T}$ et dans $\mathcal{T}'$, nous devons démontrer :

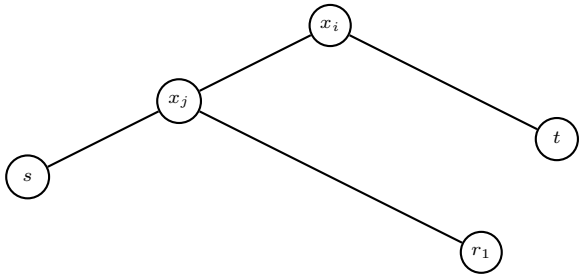
$$\max_{s,t\in\mathcal{N}(\mathcal{T})} d(s,t) = \max_{s\in\mathcal{N}(\mathcal{T})} d(s,r_1)$$

ce qui revient à

$$\forall s,t \in \mathcal{N}(\mathcal{T}),\ \exists s' \in \mathcal{N}(\mathcal{T}),\ d(s,t) \leq d(s',r_1).$$

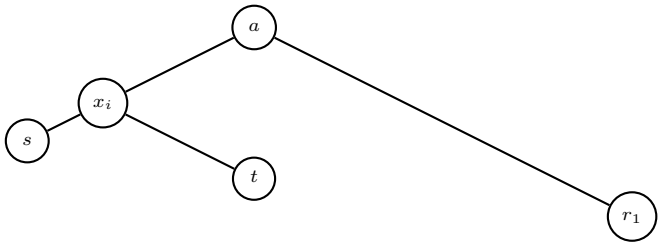
Fixons donc deux nœuds s et t de $\mathcal{T}$ et notons $(x_0,\ldots,x_k)$ le chemin de s à t et x_i le nœud le moins profond de ce chemin (x_i est le plus proche ancêtre commun à s et t). Deux cas se présentent :

- Premier cas : x_i est un ancêtre de r_1. x_i est alors ou bien le plus proche ancêtre commun à s et r_1, ou bien le plus proche ancêtre commun à t et r_1. Par symétrie, supposons que x_i est le plus proche ancêtre commun à t et r_1. Cette situation peut se schématiser de la façon suivante :



  Comme r_1 est plus profond que s dans l'arbre enraciné en x_i, nous avons $d(t,r_1) = d(t,x_i) + d(x_i,r_1) \geq d(t,x_i) + d(x_j,s) = d(s,t)$.

- Deuxième cas : x_i n'est pas un ancêtre de r_1. En notant a le plus proche ancêtre commun à x_i et r_1, nous obtenons le schéma :



On a cette fois $d(t, r_1) = d(t, a) + d(a, r_1) \geq d(t, a) + d(a, s) \geq d(t, x_i) + 1 + d(x_i, s) = d(s, t)$.

Q15 D'après les questions précédentes :

```
let diametre t = profondeur (pivot t (noeud_externe_max t));;
```

Q16 Soit r un nœud de $\mathcal{T}$ et $\mathcal{T}' = pivot(\mathcal{T}, r)$. On a clairement :

$$\forall\, s, t \in \mathcal{N}(\mathcal{T}'),\ \begin{cases} p(s, \mathcal{T}') = d(s, r) \leq D \\ d(s, t) \geq p(s, \mathcal{T}') + p(t, \mathcal{T}') \geq 2p(\mathcal{T}') \end{cases}$$

d'où $p(\mathcal{T}') \leq D \leq 2p(\mathcal{T}')$, i.e. $\left\lceil \dfrac{D}{2} \right\rceil \leq p(\mathcal{T}) \leq D$.

En choisissant $r = r_1$, $p(\mathcal{T}') = D$.

Posons $k = \lceil D/2 \rceil$ et fixons deux sommets s et t de $\mathcal{T}$ tels que $chemin(s, t) = (x_0, x_1, \ldots, x_D)$ soit un chemin de longueur maximale dans $\mathcal{T}$. En choisissant $r = x_k$, l'arbre $\mathcal{T}'$ est de profondeur k. En effet, $p(x_0, \mathcal{T}') = k$ et si s est un nœud de $\mathcal{T}'$, r est le plus proche ancêtre commun ou bien du couple (x_0, s), ou bien du couple (s, x_D).
Dans le premier cas, $D \geq d(x_0, s) = k + p(s, \mathcal{T}')$, puis $p(s, \mathcal{T}') \leq D - k \leq k$.
Dans le second cas, $D \geq d(s, x_D) = p(s, \mathcal{T}') + D - k$, puis $p(s, \mathcal{T}') \leq k$.

Nous en déduisons :

$$\max_{r \in \mathcal{N}(\mathcal{T})} p(pivot(r, \mathcal{T})) = D \text{ et } \min_{r \in \mathcal{N}(\mathcal{T})} p(pivot(r, \mathcal{T})) = \left\lceil \frac{D}{2} \right\rceil.$$

Partie III. Diffusion dans les arbres

Q17 Notons t_k la durée de diffusion dans l'arbre binomial $\mathcal{B}_k$ pour la numérotation naturelle. Le nœud r_i reçoit l'information à l'instant i et l'arbre enraciné en r_i est $\mathcal{B}_{i-1}$. La suite (t_k) vérifie donc la récurrence :

$$\begin{cases} t_0 = 0 \\ \forall k \geq 1,\ t_k = \max\limits_{1 \leq i \leq k} (i + t_{i-1}) = k + t_{k-1} \end{cases}$$

car la suite (t_k) est croissante. On en déduit donc :

$$\forall k \in \mathbb{N},\ t_k = \sum_{i=0}^{k} i = \frac{k(k+1)}{2}$$

Q18 Notons t'_k la durée de diffusion dans l'arbre binomial $\mathcal{B}_k$ pour la numérotation renversée. Nous avons cette fois :

$$\begin{cases} t'_0 = 0 \\ \forall k \geq 1,\ t'_k = \max\limits_{1 \leq i \leq k} \left(k - i + 1 + t'_{i-1}\right) \end{cases}$$

On obtient facilement $t'(k) = k$ pour $k = 0, 1, 2, 3$ et on prouve par récurrence immédiate que l'égalité s'étend à $k \in \mathbb{N}$.

Q19 $\mathcal{B}_k$ est de profondeur k : soit $(x_0, x_1, \ldots, x_k)$ une branche de $\mathcal{B}_k$. Pour une diffusion D quelconque de $\mathcal{B}_k$, nous avons $0 = t_D(x_0) < t_D(x_1) < \ldots < t_D(x_k)$ donc $t(D) \geq t_D(x_k) \geq k$. On en déduit que la durée de diffusion optimale dans $\mathcal{B}_k$ est égale à k (elle est obtenue par numérotation inversée).

Q20 L'idée (naturelle) qui se dégage de l'exemple précédent est qu'un nœud doit envoyer l'information à ses fils dans l'ordre de "complexité" décroissante : en notant $t_{opt}(\mathcal{T})$ la durée de transmission optimale d'un arbre $\mathcal{T}$, nous pouvons écrire une fonction récursive qui calcule une diffusion optimale sur un arbre $\mathcal{T}$ et qui renvoit simultanément la durée de cette diffusion optimale :

- si $\mathcal{T}$ est réduit à sa racine, il n'y a rien à construire et $t_{opt}(T) = 0$;
- si $T = (r, (\mathcal{T}_1, \ldots, T_k))$, on applique récursivement la fonction pour construire des diffusions optimales sur chaque $\mathcal{T}_i$ et calculer les durées optimales $t_i = t_{opt}(\mathcal{T}_i)$. On classe ces durées optimales :

  $$t_{\sigma(1)} \geq t_{\sigma(2)} \geq \ldots \geq t_{\sigma(k)}$$

  et on pose $f_r(i) = \sigma^{-1}(i)$. On obtient ainsi une diffusion optimale dans $\mathcal{T}$, dont la durée optimale est le maximum des valeurs $i + t_i$.

Le temps de calcul $T(\mathcal{T})$ vérifie :

$$T((r, (\mathcal{T}_1, \ldots, T_k)) = O(k \ln k) + \sum_{1 \leq i \leq k} T(\mathcal{T}_i)$$

puisqu'il faut trier un ensemble de k valeurs (la définition de f_r et le calcul du maximum des $i + D_i$ prend un temps $O(k)$). En admettant que k ne prend que des "petites" valeurs, cette récurrence se simplifie en

$$\mathcal{T}((r, (\mathcal{T}_1, \ldots, T_k)) = O(1) + \sum_{1 \leq i \leq k} T(\mathcal{T}_i)$$

ce qui donne un temps de calcul en $O(nbn(\mathcal{T}))$. Il me semble difficile d'aboutir à un temps $O(nbn(\mathcal{T}))$ en toute généralité, car si la racine possède un nombre de fils de l'ordre de grandeur de $nbn(\mathcal{T}) = n$, le tri des $t_{opt}(\mathcal{T}_i)$ demandera un temps de l'ordre de $n \ln n$, qui sera déjà prépondérant devant n.

Q21 La question est évidemment plus simple que la précédente, puisque l'on se contente de calculer la durée optimale. Nous utilisons deux fonctions auxiliaires :

- `map` f $[x_1, \ldots, x_k]$ renvoie la liste $[f(x_1); \ldots; f(x_k)]$:

```
let rec map f = function
    [] -> []
   |x::q -> (f x)::(map f q);;
```

- `trier : int list -> int list`, qui trie dans l'ordre décroissant une liste d'entiers. Une écriture naïve (en temps quadratique) serait :

```
let rec inserer x = function
   [] -> [x]
   |y::q -> if x>=y then x::y::q else y::(inserer x q);;
```

```
let rec trier = function
   [] -> []
   |x::q -> inserer x (trier q);;
```

Cela donne :

```
let rec diffusion_optimale t = match t with
   Noeud(_,[]) -> 0
   |Noeud(_,fils) -> let l=map diffusion_optimale fils in
         let M = ref 0 and i=ref 1 and p = ref (trier l) in
            while !p <> [] do
               M := max (!M) ((!i)+hd(!p));
               i := (!i)+1;
               p := tl(!p);
            done;
            (!M);;
```

Q22 Si D est une diffusion optimale pour un arbre $\mathcal{T}$, pour tout i compris entre 0 et $t(D)$, il existe un nœud v de d tel que $t_D(v) = i$. On en déduit que $\mathcal{T}$ contient au moins $t(D)+1$ nœuds distincts, soit $t_{opt}(\mathcal{T}) \leq n-1$. Cette durée est effectivement atteinte pour un arbre "peigne", i.e. pour un arbre formé par une unique branche.

Q23 Montrons par récurrence sur le nombre de nœuds l'inégalité $nbn(\mathcal{T}) \leq 2^{t_{opt}(\mathcal{T})}$.

- La propriété est évidente si $\mathcal{T}$ est réduit à sa racine ;
- Soit $n \geq 1$ et supposons que la propriété est vérifiée pour tout arbre contenant au plus $n-1$ nœuds. Soit $\mathcal{T} = (r, (\mathcal{T}_1, \ldots, T_k))$ un arbre contenant n nœuds et notons $t_0 = t_{opt}(\mathcal{T})$. Quitte à réordonner les fils de r, nous pouvons supposer qu'un diffusion optimale f vérifie $f_r(1) = 1, f_r(2) = 2, \ldots, f_r(k) = k$. Nous en déduisons que $t_{opt}(\mathcal{T}_i) \leq t_0 - i$ pour tout i compris entre 1 et k (et en particulier que $k \leq t_0$). Nous avons alors :

$$nbn(\mathcal{T}) = 1 + \sum_{i=1}^{k} nbn(\mathcal{T}_i) \leq 1 + \sum_{i=1}^{k} 2^{t_{opt}(\mathcal{T}_i)} \leq 1 + \sum_{i=1}^{t_0} 2^{t_0 - i} \leq 1 + \sum_{j=0}^{t_0-1} 2^j = 2^{t_0}$$

  et la propriété est vérifiée au rang n.

On en déduit que pour tout arbre $\mathcal{T}$ à n nœuds, $t_{opt}(\mathcal{T}) \geq \lceil \ln_2 n \rceil$.

Pour n entier naturel quelconque, soit $k = \lceil \ln_2 n \rceil$. L'arbre $\mathcal{B}_k = (r, (\mathcal{B}_{k-1}, \ldots \mathcal{B}_0))$ contient 2^k nœuds. Comme $2^{k-1} \leq n - 1 < 2^k$, on peut écrire en base 2 :

$$n - 1 - 2^{k-1} = \sum_{1 \leq i \leq d} 2^{n_i} \text{ avec } k - 2 \geq n_1 > n_2 > \ldots > n_d \geq 0.$$

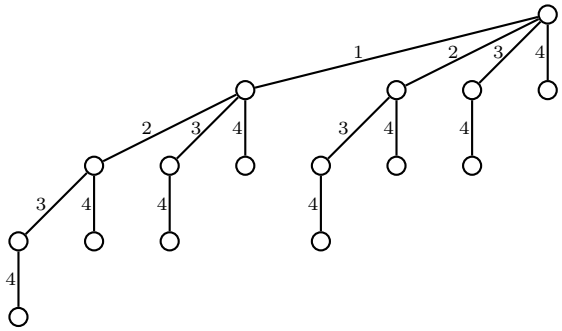
L'arbre $\mathcal{T} = (r, (\mathcal{B}_{k-1}, \mathcal{B}_{n_1}, \mathcal{B}_{n_2}, \ldots \mathcal{B}_{n_d}))$ contient alors n nœuds et $t_{opt}(\mathcal{T}) = k$. La borne obtenue précédemment est donc optimale.

Partie IV. Échange total dans les arbres

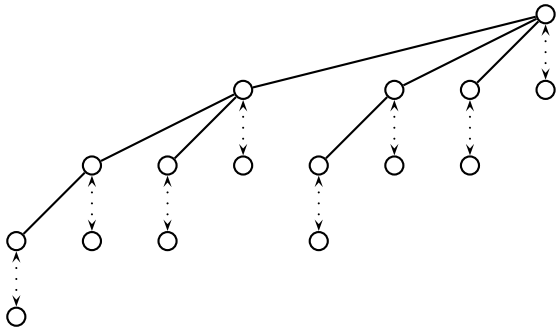
Q24 D'après la partie précédente, il est possible de diffuser l'information de la racine à tous les nœuds de $\mathcal{B}_k$ en k étapes. En effectuant ces k étapes dans l'ordre inverse, il est donc possible de remonter l'information

contenue dans tous les nœuds à la racine en k étape. La racine et son fils le plus à gauche contiennent alors toute l'information : il suffit de $k-1$ étapes pour diffuser cette information dans l'arbre entier : l'échange total dans $\mathcal{B}_k$ peut donc se faire en $2k-1$ étapes.

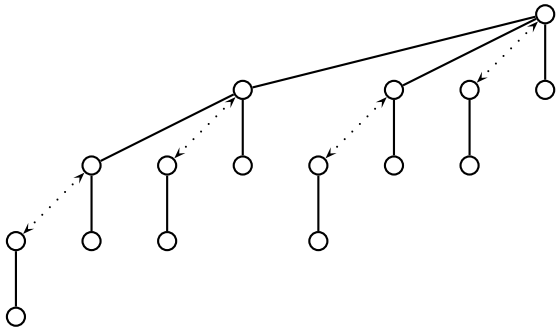
Pour détailler cette opération, prenons l'exemple de l'arbre $\mathcal{B}_4$, sur lequel une diffusion optimale est définie par le graphe :



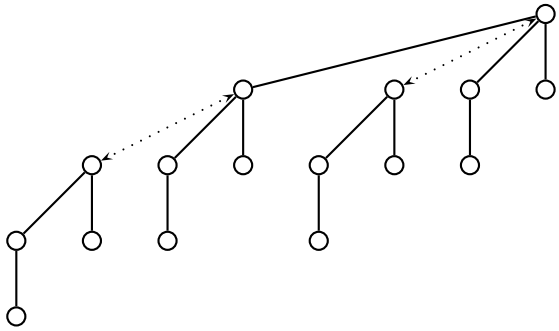
Nous commencerons donc par effectuer les 4 étapes :



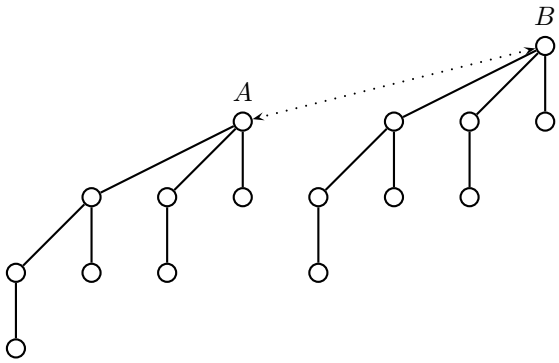
Étape 1



Étape 2



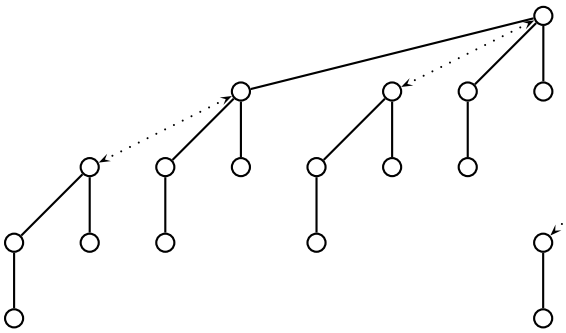
Étape 3



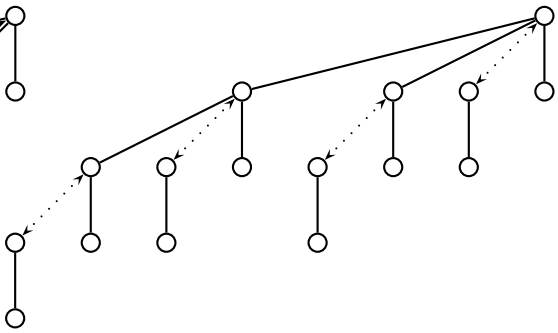
Étape 4

Les nœuds A et B connaissent alors les messages de tous les nœuds et il reste à diffuser cette information

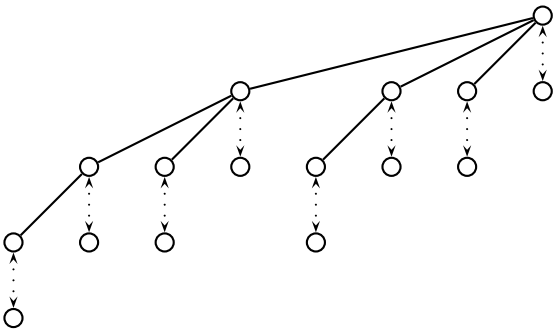
vers les autres nœuds (diffusion “parallèle” dans deux arbres binomiaux d’ordre 3) :



Étape 5



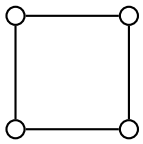
Étape 6



Étape 7

Q25 $\mathcal{B}_k$ est de diamètre $2k-1$: il existe donc deux nœuds s, t de $\mathcal{B}_k$ à la distance $2k-1$. Il faudra donc au moins $(2k-1)$ étapes pour que l’information msg_s arrive au nœud t, ce qui prouve que tout échange total dans $\mathcal{B}_k$ a une durée au moins égale à $2k-1$ (plus généralement, tout échange total dans un arbre $\mathcal{T}$ a une durée au moins égale au diamètre de $\mathcal{T}$).

Q26 L’énoncé parle “d’une” borne inférieure : devons nous comprendre “la” borne inférieure ou “un” minorant ? Soit e la durée d’un échange total dans $\mathcal{T}$. À la fin de l’étape i, un nœud ne peut connaître plus de 2^i message : on en déduit que $n \leq 2^e$, soit $e \geq \lceil \ln_2 n \rceil$. On se contentera ici de cette minoration qui n’est malheureusement pas optimale. En effet, quand $n = 4$, pour avoir $e = 2$, il faudrait que $\mathcal{T}$ contienne un cycle de la forme



et $\mathcal{T}$ ne serait pas un arbre. On a peut-être $e \geq \lceil \ln_2 2n \rceil - 1$ (ou quelque chose de cette forme), l’arbre binomial donnant alors le cas d’égalité.

Q27 Chaque passage dans la première boucle supprime un nœud de l'arbre ; comme il reste deux nœuds au sortir de cette boucle, l'instruction (1.) effectue $n-2$ échanges d'information. L'instruction 2 fait un échange et l'instruction 3 fait également $n-2$ échanges : le trafic de l'échange total est bien égal à $2n-3$.

Q28 La méthode utilisée est la suivante :

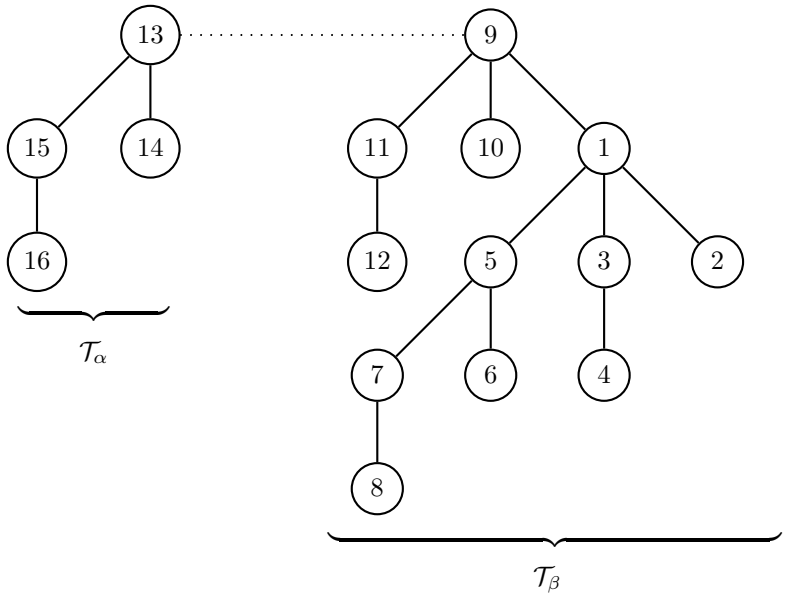
- on crée une pile `p` initialement vide ;
- on copie dans `p` la liste des nœuds de l'arbre $\mathcal{T}$ en effectuant un parcours post-fixé, par le biais de la fonction auxiliaire `remplir_pile` ;
- on renvoie la liste pointée par `p`, après avoir supprimé ses deux premiers éléments (cette liste contient les éléments demandés dans l'ordre inverse, i.e. dans l'ordre permettant d'effectuer l'étape 3 de l'échange total).

```
let rec mapbis f = function
      [] -> ()
      |t1::q -> f t1; mapbis f q;;

let echange_total t = let p = ref [] in
   let rec remplir_pile = function
      Noeud(a,fils) -> mapbis remplir_pile fils; p := a::(!p) in
   remplir_pile t;
   tl (tl (!p));;
```

Q29 Soit un échange total dans $\mathcal{T}$ de trafic égal à N. On peut représenter l'échange par une suite de N arêtes $(\{a_i, b_i\})_{1\leq i\leq N}$ de l'arbre $\mathcal{T}$ qui correspondent aux différents échanges entre voisins. Nous dirons que l'arête $\{a_i, b_i\}$ est *utilisée* à l'instant i.

Au cours de ces échanges, il existe un premier instant i_0 pour lequel un des nœuds "connaît" tous les messages. Soit $\{a_{i_0}, b_{i_0}\} = \{\alpha, \beta\}$. Ainsi, les nœuds α et β sont les seuls nœuds à posséder toute l'information à l'instant i_0. Si nous supprimons l'arête (α, β) de $\mathcal{T}$, nous séparons le graphe $\mathcal{T}$ en deux arbres $\mathcal{T}_\alpha$ et $\mathcal{T}_\beta$. Voici la construction de ces arbres sur le cas particulier $\mathcal{T} = \mathcal{B}_4$, $\alpha = 13$ et $\beta = 9$:



À l'instant $i_0 - 1$, le nœud α connaît toutes les informations des nœuds de $\mathcal{T}_\alpha$: chaque arête de $\mathcal{T}_\alpha$ a donc été utilisée au moins une fois entre l'instant 1 et l'instant $i_0 - 1$ (si γ est un nœud de $\mathcal{T}_\alpha$ distinct de α, l'échange doit nécessairement utiliser l'arête liant γ à son père pour que l'information msg_γ arrive à la racine α). De façon symétrique, c'est également le cas pour les arêtes de $\mathcal{T}_\beta$. Comme le nombre d'arête d'un arbre est égal à son nombre de nœuds diminué de 1, nous avons :

$$i_0 - 1 \geq nbn(\mathcal{T}_\alpha) - 1 + nbn(\mathcal{T}_\beta) - 1 = n - 2$$

À l'instant $i_0 - 1$, le nœud α ne connaît pas au moins une information msg_γ, où γ est un nœud de $\mathcal{T}_\beta$. On en déduit qu'à l'instant i_0, aucun nœud de $\mathcal{T}_\alpha$ autre que α ne connaît l'information msg_γ. Comme précédemment, chaque arête de $\mathcal{T}_\alpha$ est utilisée au moins une fois entre l'instant $i_0 + 1$ et N. De façon symétrique, c'est également le cas des arêtes de $\mathcal{T}_\beta$, ce qui donne :

$$N - i_0 \geq nbn(\mathcal{T}_\alpha) - 1 + nbn(\mathcal{T}_\beta) - 1 = n - 2$$

Nous obtenons ainsi

$$N = i_0 - 1 + 1 + N - i_0 \geq 2n - 3$$

Les deux dernières questions prouvent que le trafic optimal d'un échange total dans un arbre $\mathcal{T}$ est égal à $2\,nbn(\mathcal{T}) - 3$.