Corrigé DS2, sujet centrale

Q 1.

```
let nombre_aretes g = let l=ref 0 in
   for k=0 to Array.length g -1 do
      l:= !l + List.length g.(k)
   done;
   !l/2
```

Q 2.

```
let adj=[| [|1;3|] ; [|0;4;2|] ; [|1;5|] ; [|0;4|] ; [|3;1;5|] ; [|4;2|] |];;
```

Q 3.

```
let adjacence g = let n=Array.length g in let adj=Array.make n [| |] in
   for k=0 to n-1 do
      adj.(k)<-Array.of_list g.(k)
   done;
   adj;;
```

La première instruction de création de tableau est en $O(n)$. Pour chaque sommet k, l'instruction de construction de tableau à l'aide de la liste d'adjacence de k est en $O(|g(k)|)$ où $|g(k)|$ est le nombre de voisins de k. Ainsi, la boucle "for" est en $O(\sum_{k=0}^{n-1}|g(k)|)$; comme $m=\sum_{k=0}^{n-1}|g(k)|$, cette boucle est en $O(m)$. La complexité totale est donc $O(n)+O(m)=O(n+m)$.

Q 4.

```
let rang (p,q) (s,t) = if t=s+1 then p*(q-1)+s-s/p else (s mod p)*(q-1)+s/p;;
```

Q 5.

```
let sommets (p,q) i= let r=p*(q-1) in if i>=r then let s=i-r+(i-r)/(p-1) in (s,s+1) else let s=i/(q-1)+p*(i mod(q-1)) in (s,s+p);;
```

Q 6.

```
let quadrillage p q = let g=Array.make (p*q) [] in
   for i=0 to p*(q-1)+q*(p-1)-1 do
      let (s,t)=sommets (p,q) i in g.(s)<- t:: g.(s); g.(t)<- s:: g.(t)
   done;
   g;;
```

Q 7. Soit $s\in S_n$. Alors le chemin vide est un chemin de s à s donc $s\in C_s$ donc $C_s\neq\emptyset$.
Soit $s\in S_n$. Alors $s\in C_s$ donc $S_n\subset\bigcup_{s\in S_n}C_s$. Par définition, pour tout $s\in S_n$, $C_s\subset S_n$ donc $\bigcup_{s\in S_n}C_s\subset S_n$. On a donc $S_n=\bigcup_{s\in S_n}C_s$.
Soient $(s,t)\in S_n^2$. Supposons que $C_s\cap C_t\neq\emptyset$. Soit $x\in C_s\cap C_t$. Alors $x\in C_s$ donc il existe un chemin c_1 de s à x ; de même il existe un chemin c_2 de t à x. Soit $y\in C_s$. Alors il existe un chemin c_3 de s à y. Alors en juxtaposant le chemin c_2, le chemin inverse de c_1 et le chemin c_3, on obtient un chemin de t à y. Ainsi $y\in C_t$. On a donc $C_s\subset C_t$. Par symétrie $C_t\subset C_s$. On a donc $C_s=C_t$. Ainsi $C_s=C_t$ ou $C_s\cap C_t=\emptyset$.
Q 8. L'ensemble des chemins de s à t est non vide. L'ensemble des distances de ces chemins est un sous-ensemble non vide de $\mathbb{N}^*$. Il possède donc un minimum. Soit un chemin minimal $(s_0,\cdots,s_{p-1})$ de longueur p. Supposons qu'il existe $i<j$ tel que $s_i=s_j$. Alors le chemin $(s_0,\cdots,s_i,s_{j+1},\cdots,s_{p-1})$ est un chemin de s à t de longueur $i+1+(p-1)-(j+1)+1=p-(j-i)<p$ ce qui contredit la minimalité de p. Par conséquent, un chemin minimal de s à t ne passe que par des sommets deux à deux distincts.
Q 9. Supposons que les extrémités de $a_k=\{s,t\}$ soient dans la même composante connexe de G_k. Alors il existe un chemin dans G_k dont les sommets sont deux à deux distincts allant de s à t. Alors ce chemin complété avec a_k constitue un cycle. Ceci contredit le fait que G soit un arbre donc acyclique. Ainsi, les extrémités de a_k sont dans deux composantes connexes différentes.
G_0 n'a aucune arête donc possède n composantes connexes qui sont des points. Supposons, pour $0\leq k\leq m-1$, que G_k possède $n-k$ composantes connexes. Notons $a_k=\{s,t\}$. $C_s\neq C_t$. Notons $G_k=C_s\cup C_t\cup\bigcup_{i\neq s,t}C_i$ les composantes connexes de G_k. Dans G_{k+1}, s et t sont dans la mêmes composantes. Les composantes de G_{k+1} sont donc $C_s\cup C_t$ et C_i pour $i\neq s,t$. Il y en a donc une de moins, à sa voir $n-k-1$. Par récurrence, on a donc G_k possède $n-k$ composantes connexes pour tout $0\leq k\leq m$. En particulier, comme G_m est connexe, $n-m=1$ donc $m=n-1$.
Q 10. Supposons que G est un arbre. Alors, par définition G est connexe et acyclique. D'après la question précédente $m=n-1$. On a donc $(i)\Rightarrow(ii)$ et $(i)\Rightarrow(iii)$.
Supposons que G est acyclique et $m=n-1$. Notons $C_1,\cdots,C_k$ les composantes connexes de G. Alors, pour tout $1\leq i\leq k$, notons n_i le nombre de sommets de C_i et m_i son nombre d'arêtes. C_i est acyclique et connexe donc est un arbre donc $m_i=n_i-1$. Or $n=\sum_{i=1}^{k}n_i$ et $m=\sum_{i=1}^{k}m_i$. En outre, $\sum_{i=1}^{k}m_i=\sum_{i=1}^{k}(n_i-1)$ donc $m=n-k$ donc $k=1$. Ainsi, G a une seule composante connexe. Etant par hypothèse acyclique, c'est un arbre. Ceci prouve $(iii)\Rightarrow(i)$.
Montrons par récurrence sur le nombre d'arête $m=\mathrm{Card}(A)$ qu'un graphe connexe $G=(S,A)$ possède un sous-graphe $G'=(S,A')$ qui est un arbre.
Si $m=0$, G n'a pas d'arête donc pas de cycle ; étant connexe c'est un arbre.
Supposons le résultat pour $m\in\mathbb{N}$. Soit $G=(S,A)$ un graphe connexe de $m+1$ arêtes. Si le graphe est acyclique, $G=(S,A)$ est acyclique et connexe donc est un arbre. Sinon, il admet un cycle $(a_0,\cdots,a_p,a_0)$. Soit $A'=A\backslash\{a_0,a_1\}$ et $G'=(S,A')$. Soit x un

sommet de S. x est relié à a_0 par un chemin dans G ; supposons que ce chemin emprunte l'arête $\{a_1, a_0\}$; celle-ci peut être remplacée par le chemin $(a_1, \cdots, a_p, a_0)$ pour former un chemin dans G'. Ainsi, tout sommet reste relié à a_0 dans G'. G' est donc connexe et a m arêtes. De par l'hypothèse de récurrence, il admet un sous-graphe $G'' = (S, A'')$ connexe et acyclique.
Ceci prouve, par récurrence, que, pour tout nombre d'arêtes m, un graphe connexe G possède un sous-graphe $G' = (S, A')$ qui est un arbre.
Supposons que G est connexe et $m = n - 1$. Soit $G' = (S, A')$ un sous-graphe qui est un arbre. On note $m' = \text{Card}(A')$. D'après la question précédente, $m' = n - 1$ donc $m' = m$; comme $A' \subset A$, $A' = A$ donc $G = G'$ et G est un arbre. Ceci prouve $(ii) \Rightarrow (i)$.
Q 11.

```
let rec representant t i = if t.(i)<0 then i else representant t t.(i);;
```

Q 12.

```
let union t i j =if t.(i)<t.(j) then (t.(j)<-i; t.(i)<-min t.(i) (t.(j)-1)) else (t.(i)<-j; t.(j)<-min t.(j) (t.(i)-1));;
```

Q 13. Notons k le nombre d'unions utilisées pour créer $\mathcal{P}$ à partir de $\mathcal{P}_n^{(0)}$. Si $k = 0$, tout entier est un représentant et a hauteur 0. On a donc, pour tout partie X, $X = \{i\}$ pour un certain i, $|X| = 1 \geq 2^0 = 2^{h(i)}$.
Supposons que $\mathcal{P}$ est obtenu par union de deux parties de $\mathcal{P}'$, $\mathcal{P}'$ ayant obtenu par k unions. Soit X une partie de $\mathcal{P}$. Si X était déjà une partie de $\mathcal{P}'$ de représentant i, la hauteur $h(i)$ de i dans $\mathcal{P}$ et $\mathcal{P}'$ est la même donc $|X| \geq 2^{h(i)}$. Supposons que X est une partie obtenue par union des parties X_1 de représentant i de hauteur $h'(i)$ et X_2 de représentant j de hauteur $h'(j)$ dans $\mathcal{P}'$. Supposons que $h'(i) \geq h'(j)$.
Alors le nouveau représentant est i de hauteur $h(i) = \max(h'(i), h'(j) + 1) \geq h'(i)$. On a donc $|X| = |X_1| + |X_2| \geq 2^{(h'(i)} + 2^{h'(j)}$; or $2^{h'(i)} + 2^{h'(j)} \geq 2^{h'(i)}$, $2^{h'(i)} + 2^{h'(j)} \geq 2^{h'(j)} + 2^{h'(j)} = 2^{1+h'(j)}$ donc $2^{h'(i)} + 2^{h'(j)} \geq 2^{\max(h'(i),h'(j)+1)}$ donc $|X| \geq 2^{h(i)}$.
Si $h(i) \leq h(j)$, on effectue comme ci-dessus en échangeant le rôle de i et de j.
Ainsi, par récurrence, on a bien, pour toute partie X de représentant i, $|X| \geq 2^{h(i)}$.
Q 14. La fonction "representant" effectue un nombre d'appels récursifs de l'ordre de $h(i)$, donc est en $O(h(i))$; d'après la question précédente, $h(i) = O(\log |X|)$ et $|X| \leq n$; la complexité de cette fonction est donc $O(\log n)$. La fonction "union" effectue deux comparaisons et deux affectations. Elle est donc en $O(1)$.
Q 15.

```
let est_un_arbre g = let n=Array.length g in let p=Array.make n -1 and composante=ref n in
   let rec aux i l = match l with
      | [] -> ()
      | j::q -> let s=representant i and t=representant j in
                if i<>j then (union p s t; composante:=!composante-1; aux s q) in
   for k=0 to n-1 do
      aux k g.(k)
   done;
   !composant=1 && nombre_aretes g=n-1;;
```

Q 16. Le chemin représenté est $(1, 2, 5, 4)$.
Q 17. Si le choix du nouveau sommet est mal fait, si on évite toujours les sommets de $\mathcal{T}$, l'algorithme ne se termine pas.
Q 18.

```
let marche_aleatoire adj parent s =
   let c={debut = s ; fin = s ; suivant = Array.make Array.length adj -1} in
   while parent.(c.fin)=-2 do
      let voisins=adj.(c.fin) in
      let n=Array.length voisins in
      let u=voisins.(Random.int n) in
      suivant.(c.fin) <- u;
      suivant.(u) <- -1;
      c.fin <- u
   done;
   c;;
```

Q 19.

```
let greffe parent c = let n=Array.length (c.suivant) and s=ref c.debut in
   while !s<>c.fin do
      parent.(!s) <- c.suivant.(!s);
      s:=c.suivant.(!s)
   done;;
```

Q 20.

```
let wilson g r = let n=Array.length g and adj=adjacence g in
   let parent=Array.make n -2 in parent.(r) <- -1;
   for s=0 to n-1 do
      if parent.(s)=-2 then let c=marche_aleatoire adj parent s in greffe parent c
   done;
   parent;;
```

Q 21.

Q 22.

Q 23. Si la direction en s est N, le père de s est au-dessus, c'est donc $s+p$; si la direction est S, le père est en-dessous, donc en $s-p$; si la direction est W, le père est à gauche donc en $s-1$; si la direction est E, le père est à droite donc en $s+1$.
Q 24.

```
let coord_noire s = ( 2*(s mod p) , 2*(s/p) );;
```

Q 25.

```
let sommet_direction s d = match d with
   | N -> if s<=(q-1)*p then s+p else -1
   | S -> if s>=p then s-p else -1
   | W -> if s mod p >0 then s-1 else -1
   | E -> if s mod p <p-1 then s+1 else -1;;
```

Q 26.

```
let phi pavage =
   let parent=Array.make p*q -1 in
   for s=1 to p*q-1 do
      let (i,j)=coord_noire s in parent.(s) <- sommet_direction s pavage.(i).(j)
   done;
   parent;;
```

Q 27.

```
let dual =
   let g=Array.make ((p-1)*(q-1)+1) [] in
   for i=0 to p-2 do
      for j=0 to q-2 do
         if i>0 && i<p-2 && j>0 && j<q-2 then let k=j*(p-1)+j in g.(k) <- [k-1;k+1;k+p-1;k-p+1];
         if i=0 && j>0 && j<q-2 then (let k=j*(p-1)+1 in g.(k) <- [0;k+1;k+p-1;k-p+1]; g.(0) <- k::g.(0));
         if i=p-2 && j>0 && j<q-1 then (let k=j*(p-1)+p-1 in g.(k) <- [k-1:0;k+p-1;k-p+1]; g.(0) <- k::g.(0));
         if i>0 && i<p-2 && j=0 then (let k=i+1 in g.(k) <- [k-1;k+1;k+p-1;0]; g.(0) <- k::g.(0));
         if i>0 && i<p-2 && j=q-2 then (let k=(q-1)*(p-1)+i+1 in g.(k) <- [k-1;k+1;0;k-p+1]; g.(0) <- k::g.(0));
         if i=0 && j=0 then (let k=1 g.(k) <- [0;k+1;k+p-1;0]; g.(0) <- k::g.(0));
         if i=p-2 && j=0 then (let k=p-1 in g.(k) <- [k-1;0;k+p-1;0]; g.(0) <- k::g.(0));
         if i=0 && j=q-2 then (let k=(q-2)*(p-1) in g.(k) <- [0;k+1;0;k-p+1]; g.(0) <- k::g.(0));
         if i=p-2 && j=q-2 then (let k=(q-2)*(p-1)+p-1 in g.(k) <- [k-1;0;0;k-p+1]; g.(0) <- k::g.(0));
      done
   done;
   g;;
```

Q 28. Le graphe $G^*_{p,q}$ délimite aussi des faces. Chacune de ces faces correspond exactement à un sommet de $G_{p,q}$. Par définition du dual, les sommets de $G^{**}_{p,q}$ sont les faces de $G^*_{p,q}$; par conséquent, les sommets de $G^{**}_{p,q}$ sont les sommets de $G_{p,q}$.
Deux faces de $G^*_{p,q}$ ont une frontière commune si et seulement si les deux sommets de $G_{p,q}$ auxquels elles correspondent sont reliés par une arête. Par définition du dual, une arête de $G^{**}_{p,q}$ correspond à une arête de $G^*_{p,q}$ qui sépare deux faces de $G^*_{p,q}$, ces deux faces correspondant à deux sommets de $G_{p,q}$. Toujours par définition du dual, cette arête de $G^*_{p,q}$ correspond à une arête de $G_{p,q}$ reliant les deux sommets en question. Par conséquent, une arête de $G^{**}_{p,q}$ correspond à une arête de $G_{p,q}$.
En conclusion, le dual de $G^*_{p,q}$ est bien $G_{p,q}$.
Q 29. Supposons que (S_n, B) possède un cycle c. Alors ce cycle sépare le plan en les sommets à l'intérieur et les sommets à l'extérieur. Tout chemin entre une face à l'intérieur de ce cycle et une face à l'extérieur de ce cycle doit passer par la frontière, à savoir par une arête a^* entre une face intérieure et une face extérieure. Cette arête a^* correspond à une arête a intervenant dans le cycle. Or B^* ne contient aucune de ces arête a^* par définition donc il ne peut y avoir de chemin entre une face intérieur et une face extérieur dans (S_{n^*}, B^*). Par conséquent, (S_{n^*}, B^*) n'est pas connexe.
Q 30. Supposons que (S_n, B) est un arbre couvrant de $G_{p,q}$. Supposons que (S_{n^*}, B^*) a un cycle. Alors son dual est (S_n, B) et, d'après la question précédente, n'est pas connexe, ce qui est absurde. Par conséquent, (S_{n^*}, B^*) est acyclique.
En outre, le nombre de sommets de S_{n^*} est $n^* = (p-1)(q-1)+1 = pq-p-q+1+1 = n-p-q+2$; B^* correspond au complémentaire de B dans les arêtes de $G_{p,q}$. Il y a $(p-1)q + p(q-1)$ arêtes dans $G_{p,q}$; notons m le nombre d'arêtes de B et m^* celui de B^*. On a donc $m^* = p(q-1) + q(p-1) - m = 2pq - p - q - m = 2n - p - q - m = n - p - q + n - m = n^* - 2 + n - m$. Comme (S_n, B) est un cycle, $n - m = 1$ donc $m^* = n^* - 2 + 1 = n^* - 1$.
D'après la question 10, le graphe (S_{n^*}, B^*) est un arbre (donc un arbre couvrant par définition de son ensemble de sommets).
Q 31.

```
let vers_couple parent = let r=ref -1 and b=Array.make (p*(q-1)+q*(p-1)) false in
   for s=0 to (Array.length parent)-1 do
      let t=parent.(s) in if t=-1 then r:=s
                          else let a = if s<t then rang (p,q) (s,t) else rang (p,q) (t,s) in b.(a) <- true
   done;
   (!r,b);;
```

Q 32. On crée un tableau `parent` de taille pq de -1. On considère toutes les arêtes `a` partant de r de type $\{r, p\}$ et on pose `parent.(p) <- r` et on effectue la même chose pour `p`. Cet algorithme peut être appliquer à $G^*_{p,q}$ à condition de disposer, comme pour $G_{p,q}$ d'une bonne bijection entre arêtes de $G^*_{p,q}$ et couples de sommets (comme le font les fonctions 4 et 5).
Q 33.

```
let vers_parent (r,b) = let parent=Array.length (p*q) -1 in
   let rec aux u =
      for i=0 to p*(q-1)+q*(p-1)-1 do
         if b.(i) then begin
            let (s,t)=sommets (p,q) i in
            if u=s then (parent.(t) <- u; aux t);
            if u=t then (parent.(s) <- u; aux s) end
      done in aux r;
   parent;;
```

Q 34. Les calculs de rang et de sommets se font en $O(1)$. Le tableau "parent" a taille pq ; le fonction "vers couple" contient une boucle de taille pq. La complexité est donc $O(pq)$.
La fonction "vers parent" repose sur une fonction récursive qui effectue un appel récursif sur chaque voisin de l'argument. Chaque appel récursif effectue une boucle de longueur $O(p(q-1)+q(p-1))$. Comme il y a pq sommets, la complexité est de $O\big(pq\times(p(q-1)+q(p-1))\big) = O(p^2q^2)$.
Q 35.

```
let arbre_dual parent = let (r,b)=vers_couple parent in
   for k=0 to Array.length b -1 do
      b.(k) <- not(b.(k))
   done;
   vers_parent_etoile (r,b);;
```

Q 36. On parcourt tous les points noirs et les points gris. Pour un tel point, on regarde le père de ce point. Si son père est au-dessus, on attribue à ce point la direction N, s'il est en-dessous, la direction S, s'il est à gauche la direction W, à droite la direction E.
Si un sommet est noir et son numéro est i, son père est au-dessus (resp. en-dessous, resp à gauche, resp. à droite) s'il s'agit de $i+p$ (resp. $i-p$, resp. $i-1$, resp. $i+1$).
Si un sommet est gris et son numéro est i, si son père est différent de 0, il est au-dessus (resp. en dessous, resp. à gauche, resp. à droite) s'il s'agit de $i+p-1$ (resp. $i-p+1$, resp. $i-1$, resp. $i+1$).
Si son père est 0, il est au-dessus si $(i-1)/(p-1) = q-1$ et $i-1 \mod (p-1) \neq 0, p-1$, il est en-dessous si $(i-1)/(p-1) = 0$ et $(i-1) \mod (p-1 \neq 0, p-1$, il est à gauche si $(i-1) \mod (p-1) = 0$ et $(i-1)/(p-1) \neq 0, q-1$, à droite si $(i-1) \mod (p-1) = p-2$ et $(i-1)/(p-1) \neq 0, q-1$.
Enfin, dans le cas d'un des quatre points formant les angles de l'échiquier, si le père est 0, il faudra regarder la direction du domino en position noir de l'angle correspondant afin d'éliminer l'une des deux directions possibles. Par exemple, pour le point gris en haut à gauche, si la direction du point noir est E (resp. S), celle du point gris sera W (resp. N).
Q 37.

```
let pavage_aleatoire = let parent=wilson g 0 in let parent_etoile=arbre_dual parent in
   let pavage=Array.make_matrix (2*p-1) (2*q-1) N in
   let aux_noir i = match i with
      | 0 -> ()
      | i when parent.(i)=i-1 -> W
      | i when parent.(i)=i+1 -> E
      | i when parent.(i)=i+p -> N
      | _ -> S
   and aux_gris i = match i with
      | 1 when parent.(i)=0 -> let (a,b)=coord_noire i in if pavage.(a).(b)=W then E else S
      | p-1 when parent.(i)=0 -> let (a,b)=coord_noire i in if pavage.(a).(b)=N then S else E
      | (q-2)*(p-1)+1 when parent.(i)=0 -> let (a,b)=coord_noire i in if pavage.(a).(b)=S then N else W
      | (q-1)*(p-1) when parent.(i)=0 -> let (a,b)=coord_noire i in if pavage.(a).(b)=S then N else W
      | i when parent.(i)=0 && (i-1)/(p-1)=0 -> S
      | i when parent.(i)=0 && (i-1)/(p-1)=q-2 -> N
      | i when parent.(i)=0 && (i-1)mod(p-1)=0 -> W
      | i when parent.(i)=0 && (i-1)mod(p-1)=p-2 -> E
      | i when parent.(i)=i+1 -> E
      | i when parent.(i)=i-1 -> W
      | i when parent.(i)=i+p-1 -> N
      | _ -> S in
   for x=0 to p-1 do
      for y=0 to q-1 do
         aux_noir (numero (2*x,2*y))
      done
   done;
   for x=0 to p-2 do
      for y=0 to q-2 do
         aux_gris (numero (2*x+1,2*y+1))
      done
   done;
   pavage;;
```

Q 38. Pour construire un pavage de taille $2p, 2q-1$, il suffit de créer un pavage aléatoire de taille $2p-1, 2q-1$ et d'ajouter une colonne de q dominos verticaux à droite de ce pavage. Pour construire un pavage de taille $2p, 2q$, il suffit de reprendre un pavage de taille $2p, 2q-1$ et d'ajouter une ligne de p dominos horizontaux au-dessus de ce pavage.