1 Étude préliminaire

1. Il suffit de supprimer les murs associés aux arêtes conservées dans l'arbre couvrant, et de garder les autres murs. Comme l'arbre est couvrant, pour toutes positions s et t dans le labyrinthe, il existe bien un chemin allant de s à t, et comme c'est un arbre, un tel chemin est unique (sinon, on aurait un cycle !).
2. On peut calculer un arbre couvrant avec une version simplifiée de l'algorithme de Kruskal : on aura toujours besoin d'une structure Union-Find, mais pas besoin de file de priorité car ici le graphe considéré n'est pas pondéré.
   Pour obtenir un labyrinthe parfait différent, il suffit de changer l'ordre dans lequel les arêtes sont considérées lors de l'algorithme de Kruskal.

2 Mélange équitable

3. Une telle fonction `melange` termine toujours avec la même complexité, mais son résultat peut varier : il s'agit d'un algorithme de type **Monte Carlo**. Il aura la propriété supplémentaire que son résultat est **toujours correct** (il y a plusieurs résultats corrects pour une même entrée).
4. - Cas $i = 0$: avant la boucle, on a bien que $x_0 = y_0^0$ car le tableau n'a pas encore été modifié, et $\mathbb{P}(0, \sigma) = 1$ pour l'unique permutation $\sigma \in \mathfrak{S}_{[\![0,0]\!]}$.
   - Soit $i > 0$, supposons la propriété vraie pour $i-1$. Soit $\sigma \in \mathfrak{S}_{[\![0,i]\!]}$.
     - si $\sigma(i) = i$, alors le seul moyen d'obtenir l'évènement (i, σ) est d'avoir tiré $j = i$ (avec probabilité $\frac{1}{i+1}$), et dans ce cas :
     $$\mathbb{P}(i,\sigma) = \underbrace{\mathbb{P}(j=i)}_{\frac{1}{i+1}} \times \underbrace{\mathbb{P}(i-1, \sigma_{|[\![0,i-1]\!]})}_{\frac{1}{i!} \text{ (par HR)}} = \frac{1}{i+1} \times \frac{1}{i!} = \frac{1}{(i+1)!}$$
     - si $\sigma(i) = k_1 < i$, alors il existe k_2 tel que $\sigma(k_2) = i$.
       Posons $\tilde{\sigma} \in \mathfrak{S}_{[\![0,i-1]\!]}$ telle que $\tilde{\sigma}(k_2) = k_1$ et $\forall k \neq k_2,\ \tilde{\sigma}(k) = \sigma(k)$.
       Pour obtenir l'évènement (i, σ), il n'y a qu'une seule possibilité :
       - à l'étape d'avant, on a eu l'évènement $(i-1, \tilde{\sigma})$;
       - à l'étape i, on a tiré $j = k_2$.

       Ainsi :
       $$\mathbb{P}(i,\sigma) = \underbrace{\mathbb{P}(j=k_2)}_{\frac{1}{i+1}} \times \underbrace{\mathbb{P}(i-1, \tilde{\sigma})}_{\frac{1}{i!} \text{ (par HR)}} = \frac{1}{i+1} \times \frac{1}{i!} = \frac{1}{(i+1)!}$$
5. Ainsi, à la fin de la dernière étape de l'algorithme ($i = n-1$), on a :
   $$\forall \sigma \in \mathfrak{S}_{[\![0,n-1]\!]},\ \mathbb{P}(n-1, \sigma) = \frac{1}{n!}$$
   Donc le mélange de Knuth est un mélange équitable.
6. ```
   let melange_knuth tab =
     let n = Array.length tab in
     for i = 1 to n-1 do
       let j = Random.int (i+1) in
       let tmp = tab.(i) in
       tab.(i) <- tab.(j) ;
       tab.(j) <- tmp
     done
   ```
   ```

3 Générateur de labyrinthes parfaits

7. Il y a $2n(n-1)$ murs internes dans une grille de taille $n \times n$.

8.
```
let generer_aretes n : arete array =
  let aretes = Array.make (n*(n-1)*2) (0,0,H) in
  let k = ref 0 in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if i < n-1 then
        begin
          aretes.(!k) <- (i,j,V) ;
          incr k
        end ;
      if j < n-1 then
        begin
          aretes.(!k) <- (i,j,H) ;
          incr k
        end
    done ;
  done ;
  aretes
```

9.
```
let labyrinthe_parfait n =
  let uf = create (n * n) in
  let horiz = Array.make_matrix n (n-1) true in
  let vert  = Array.make_matrix (n-1) n true in
  let aretes = generer_aretes n in
  melange_knuth aretes ;
  for l = 0 to Array.length aretes - 1 do
    let (i, j, hv) = aretes.(l) in
    let k = i * n + j in
    let k' = if hv = H then k + 1 else k + n in
    if find uf k <> find uf k' then
      begin
        if hv = H
        then horiz.(i).(j) <- false
        else vert.(i).(j) <- false ;
        union uf k k'
      end
  done ;
  (horiz, vert)
```