Exploration récursive

ICC-C Cours 14

searchjobs.me

Vlad Mihalcea • 2nd
Java Champion
1h •

The maximum number of nested for loops is 18 because that's the number of letters between i and z.

357 13 comments · 9 repost

Afficher un tableau

- Pourquoi toujours `tableau[0]`?
- Pourquoi `tableau + 1` ?
- Le paramètre `taille` est important
- Et si on met `taille > 5` ?
- Et si on met le `printf` après l'appel récursif?

```c
#include <stdio.h>

void afficher_ptr(int *tableau, int taille)
{
    if (taille > 0)
    {
        printf("%d ", tableau[0]);
        afficher_ptr(tableau + 1, taille - 1);
    }
    else
    {
        printf("\n");
    }
}

int main()
{
    int valeurs[] = {10, 20, 30, 40, 50};

    afficher_ptr(valeurs, 5);
}
```

Palindrome

- Appel de cette fonction ?

```
char *s = “kayak";

int longueur = strlen(s);

palindrome_ptr(
  s, s + longueur - 1);
```

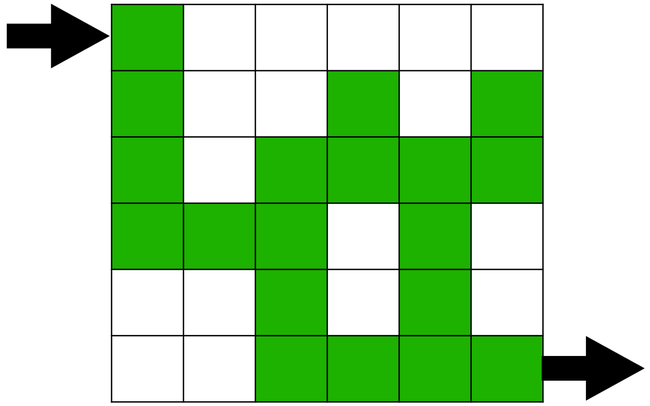
```
int palindrome_ptr(char *debut, char *fin)
{
    if (debut >= fin)
    {
        // les pointeurs se sont croisés - ok !
        return 1;
    }
    if (*debut != *fin)
    {
        // ce n’est pas un palindrome
        return 0;
    }
    return palindrome_ptr(debut + 1, fin - 1);
}
```

```
| k | a | y | a | k | '\0' | ...
  ^               ^
  debut           fin
```

Bix

… ou comment explorer un labyrinthe

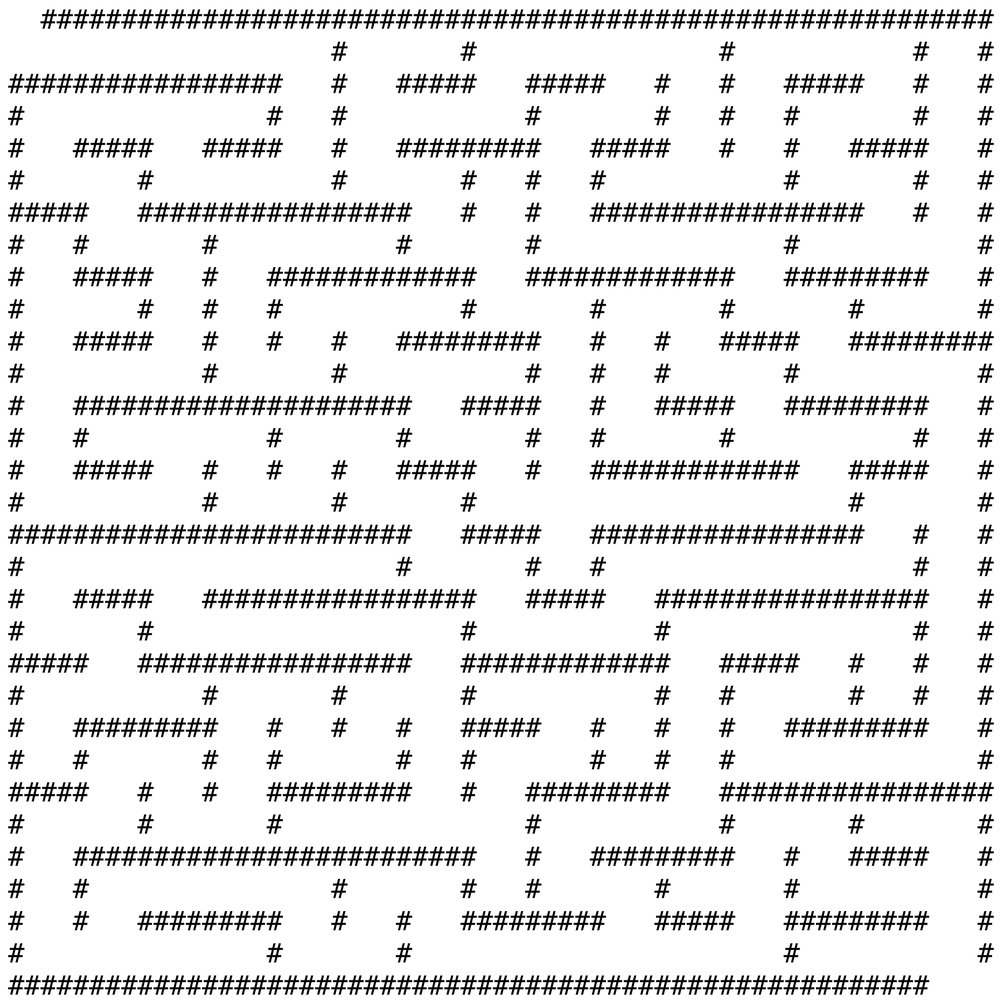
- Représentation du labyrinthe



```
1 0 0 0 0 0
1 0 0 1 0 1
1 0 1 1 1 1
1 1 1 0 1 0
0 0 1 0 1 0
0 0 1 1 1 1
```

Bix

Un peu plus compliqué



=

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0
0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0
0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0
0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0
0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0
0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 0
0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0
0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 0
0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0
0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0
0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0
0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0
0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0
0 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0
0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
```

Bix

Est-ce qu'il existe une solution?

```
int explorer(int r, int c)
{
    if (r < 0 || c < 0 ||
        r >= M || c >= N)
    {
        return 0;
    }

    if (map[r][c] == 0)
    {
        return 0;
    }

    if (r == M - 1 && c == N - 1)
    {
        return 1;
    }

    if (explorer(r - 1, c))
    {
        return 1;
    }

    if (explorer(r + 1, c))
    {
        return 1;
    }

    if (explorer(r, c + 1))
    {
        return 1;
    }

    if (explorer(r, c - 1))
    {
        return 1;
    }

    return 0;
}
```

❌

☠️

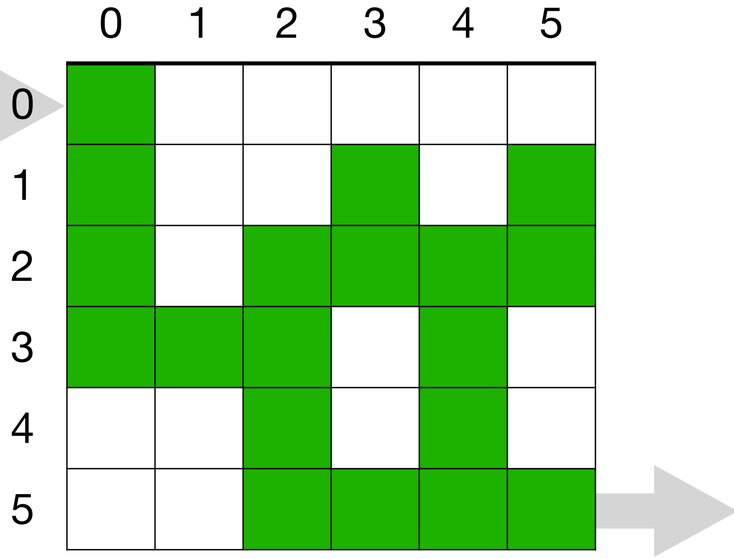
🏁

haut

bas

droite

gauche




```
0: explorer(0, 0)
├─1(h): explorer(-1, 0)
│  └─ ❌
└─2(b): explorer(1, 0)
   └─3(h): explorer(0, 0)
      ├─4(h): explorer(-1, 0)
      │  └─ ❌
      └─5(b): explorer(1, 0)
```

∞

Bix

… et le Petit Poucet

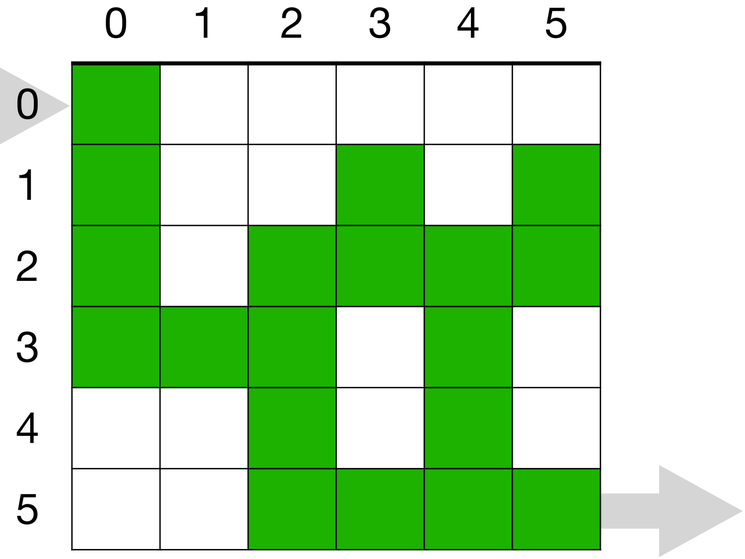
- Tableau 2-d `vu[i][j]`
  = je suis déjà passé par la case `(i, j)`

```
int explorer(int r, int c)
{
    if (r < 0 || c < 0 ||
        r >= M || c >= N)
    {
        return 0;
    }

    if (map[r][c] == 0)
    {
        return 0;
    }

    if (vu[r][c])
    {
        // Nous sommes déjà passés par là
        return 0;
    }

    // Marquer comme vu avant de commencer
    // l'exploration des voisins
    vu[r][c] = 1;
```

❌ ☠️ 👁️

```
    if (r == M - 1 && c == N - 1)
    {
        return 1;
    }

    if (explorer(r - 1, c))
    {
        return 1;
    }

    if (explorer(r + 1, c))
    {
        return 1;
    }

    if (explorer(r, c + 1))
    {
        return 1;
    }

    if (explorer(r, c - 1))
    {
        return 1;
    }

    return 0;
}
```

🏁 haut bas droite gauche

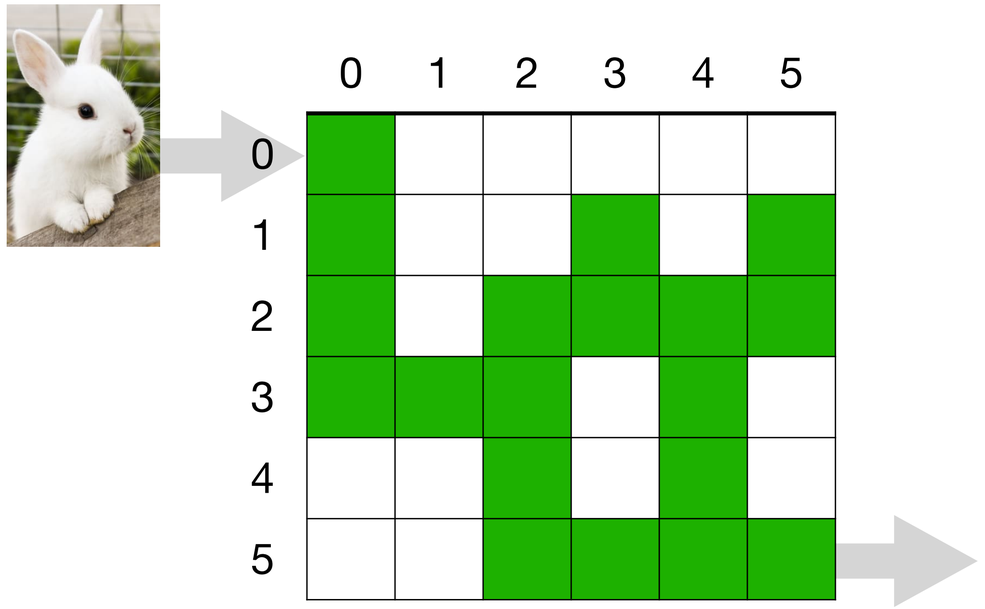



```
0: explorer(0, 0)
├─1(h): explorer(-1, 0)
│  └─ ❌
└─2(b): explorer(1, 0)
   ├─3(h): explorer(0, 0)
   │  └─ 👁️
   └─4(b): explorer(2, 0)
...
```

✅

Bix

Est-ce qu’il existe une solution?



- Oui! 🎉
- Comment peut-on la retrouver?
- “bbbddbbddd”
- Nous pouvons sauvegarder des résultats intermédiaires dans un accumulateur…

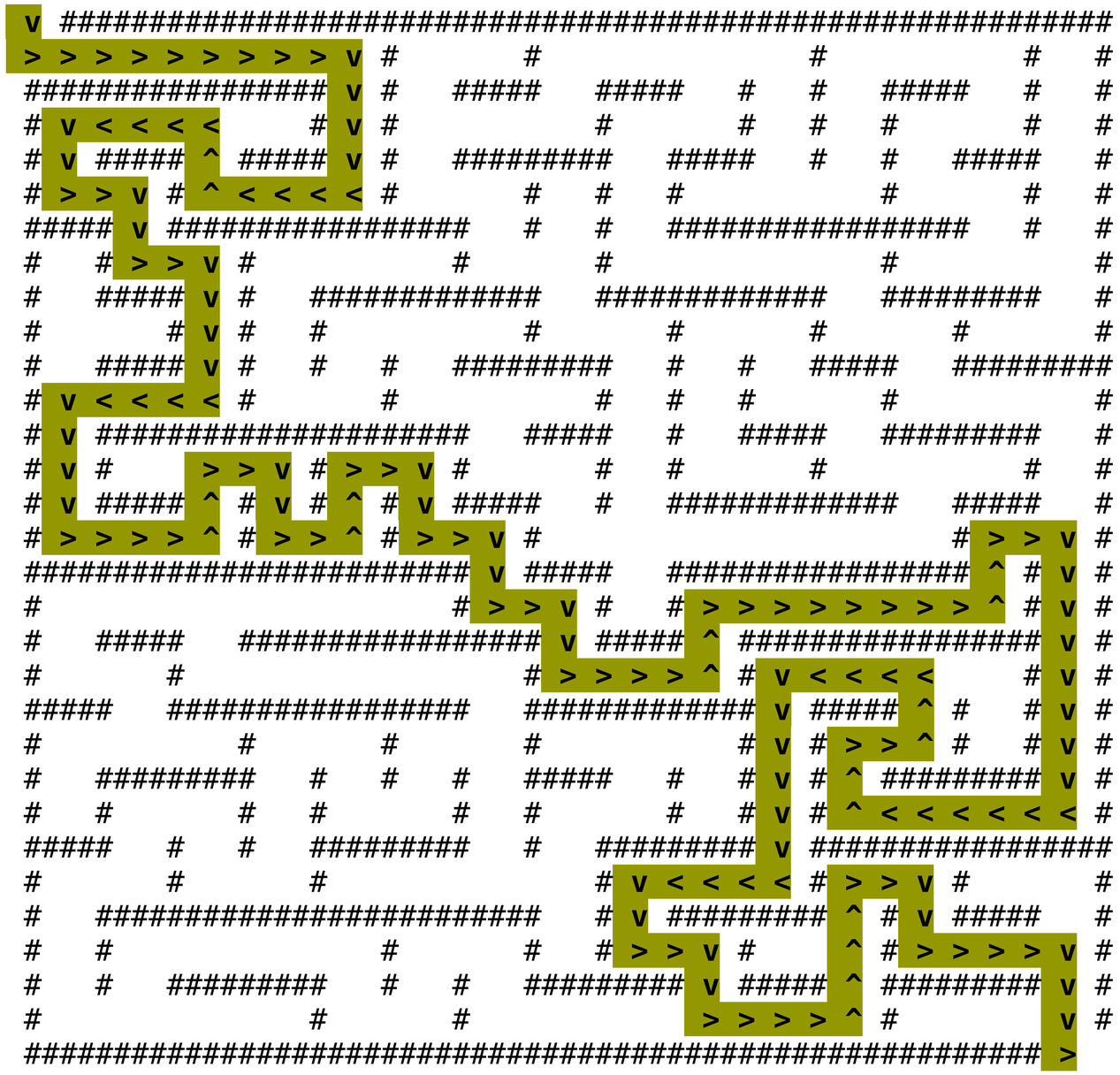
Sauvegarder les choix

```
int explorer2(int r, int c,
              char *instructions, int indice)
{
    ...
    if (r == M - 1 && c == N - 1)
    {
        // Nous avons trouvé la destination !
        instructions[indice] = '\0';
        return 1;
    }

    instructions[indice] = 'h';
    if (explorer2(r - 1, c, instructions, indice + 1))
    {
        return 1;
    }

    instructions[indice] = 'b';
    if (explorer2(r + 1, c, instructions, indice + 1))
    {
        return 1;
    }

    instructions[indice] = 'd';
    if (explorer2(r, c + 1, instructions, indice + 1))
    {
        return 1;
    }

    instructions[indice] = 'g';
    if (explorer2(r, c - 1, instructions, indice + 1))
    {
        return 1;
    }
    return 0;
}
```

```
char instructions[1000];

if (explorer2(0, 0, instructions, 0))
{
    printf("Chemin trouvé : %s\n", instructions);
}
```

Réponse

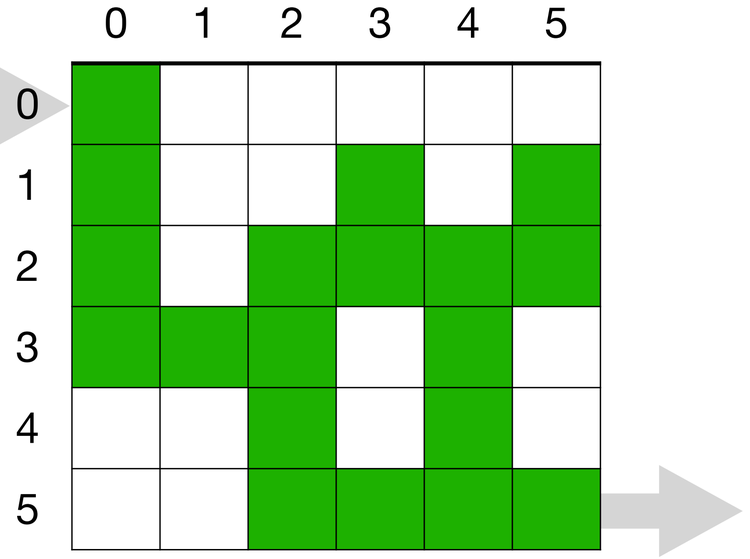
bddddddddbbbbgggghhggggbbd
dbbddbbbbggggbbbbddddhhddbb
ddhhddbbddbbddbbddddhhdddddd
dddhhddbbbbbbbbgggggghhddhh
ggggbbbbbbggggbbddbbddddhhh
hddbbddddbbbd

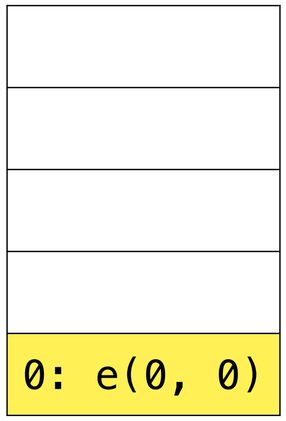


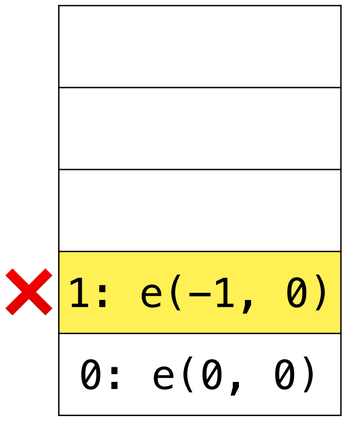
Trouver un chemin de sortie

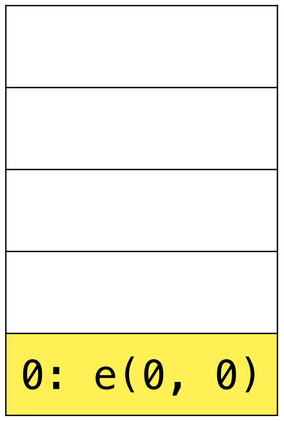
- Algorithme **DFS** = Depth-First Search
- Ce n’est pas la seule manière, par exemple: **BFS** = Breadth-First Search
  - **BFS** nous permet en plus de trouver le chemin de sortie **le plus court**
- Nous venons de voir une implémentation récursive de **DFS**
- Implémentation **itérative** — nous avons besoin d’une **pile**!

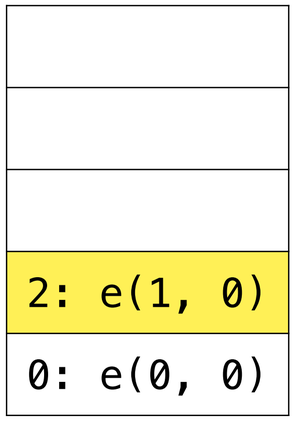
Pile

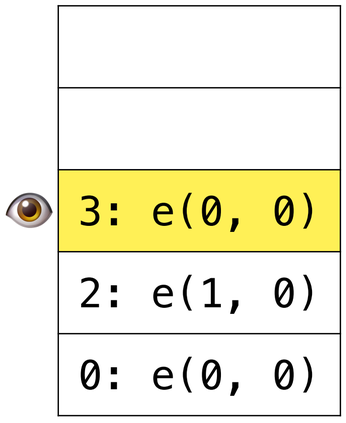


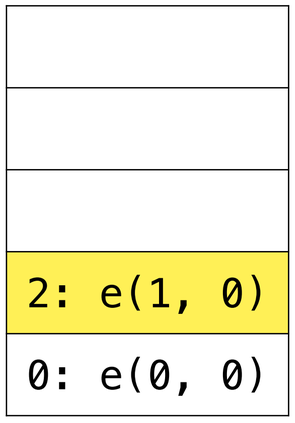


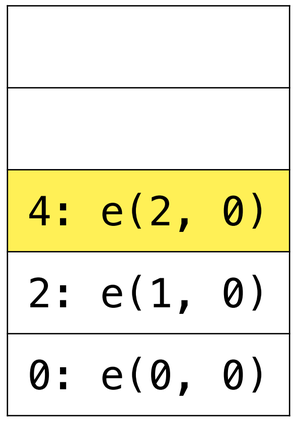













```
0: explorer(0, 0)
├─1(h): explorer(-1, 0)
│  └─ ❌
└─2(b): explorer(1, 0)
   ├─3(h): explorer(0, 0)
   │  └─ 👁
   └─4(b): explorer(2, 0)
...
```

✅

Générer les permutations

- On aimerait calculer/afficher toutes les permutations d'une certaine taille `N`
- Il y en a `N!`

- Exemple `N = 3`:

```
0, 1, 2

0, 2, 1

1, 0, 2

1, 2, 0

2, 0, 1

2, 1, 0
```

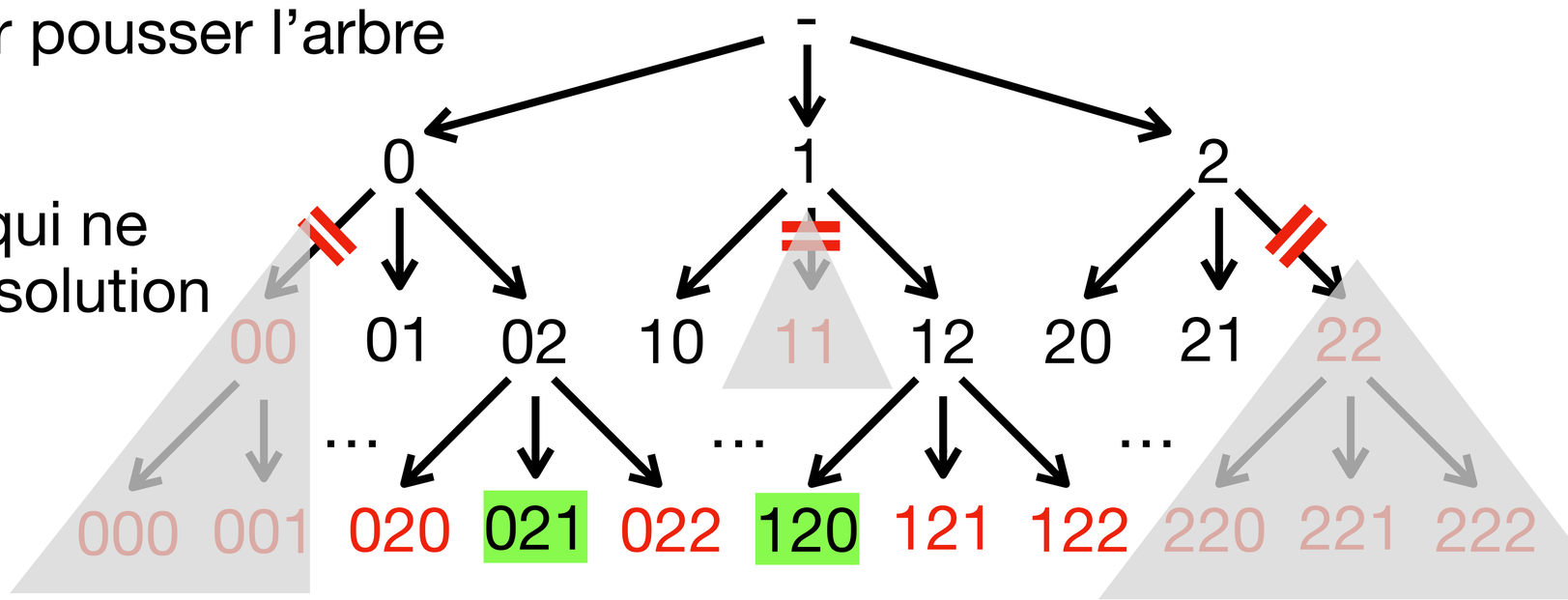
For-For-For

```c
int perm[3];

for (int i = 0; i < 3; i++)
{
    perm[0] = i;
    for (int j = 0; j < 3; j++)
    {
        perm[1] = j;
        for (int k = 0; k < 3; k++)
        {
            perm[2] = k;
            if (est_valide_p3(perm))
            {
                printf("%d %d %d\n",
                       perm[0],
                       perm[1],
                       perm[2]);
            }
        }
    }
}
```

```c
int est_valide_p3(int *perm)
{
    int vu[3];
    memset(vu, 0, 3 * sizeof(int));

    for (int i = 0; i < 3; i++)
    {
        vu[perm[i]] = 1;
    }

    int valide = 1;
    for (int ell = 0; ell < 3; ell++)
    {
        if (!vu[ell])
        {
            valide = 0;
            break;
        }
    }
    return valide;
}
```

Qu’est-ce que ça fait?

- Le code précédent naïvement explore toutes les possibilités (3^3 = 27)
- On construit un **arbre de recherche** et on choisit les bonnes **feuilles**
- En fait il ne faut pas laisser pousser l’arbre dans tous les sens…
- Il faut tailler les branches qui ne peuvent pas mener à une solution



Généraliser

- Comment fait-on pour écrire N boucles `for` imbriquées ??
- **Réponse possible**: On écrit une fonction récursive!
- Elle aura un cas de base (quand on atteint N)
- Faudra vérifier que c’est une permutation valide le plus tôt possible
- Relation de récurrence pour construire la solution

Générer les permutations

```
void enumerer_pn(int *perm, int *vu, int indice, int n)
{
    if (indice == n)
    {
        afficher_pn(perm, n);
        return;
    }

    for (int valeur = 0; valeur < n; valeur++)
    {
        if (!vu[valeur])
        {
            vu[valeur] = 1;
            perm[indice] = valeur;
            enumerer_pn(perm, vu, indice + 1, n);
            vu[valeur] = 0;
        }
    }
}
```

Le cas de base

Elaguer les branches!

Relation de récurrence
Exploration

Backtracking

Recette générique

- On aimerait itérer sur des objets d'une certaine taille N (e.g., permutations)
- Attention — ça coûte **cher** (souvent temps exponentiel!) ~ "force brute"
- Pourquoi?
  - On veut en trouver un objet valide/le meilleur
  - On ne connait pas d'algorithme plus intelligent
- Un objet a N composants/dimensions
- On explore les valeurs possibles de chaque dimension
- On a une méthode pour évaluer si on peut **arrêter tôt** l'exploration d'une branche

Backtracking

Recette générique

- Existe aussi en itératif
- Faute d'une meilleure méthode…

```
void enumerer(T *objet, int *vu, int indice, int n)
{
    if (indice == n)
    {
        if (est_valide(objet, n))
        {
            // faire qqch avec cet objet
        }
        return;
    }

    for (int i = 0; i < n_valeurs; i++)
    {
        if (!vu[i])
        {
            vu[i] = 1;
            objet[indice] = valeur[i];
            enumerer(objet, vu, indice + 1, n);
            vu[i] = 0;
        }
    }
}
```

Nombre de valeurs possibles d'une dimension

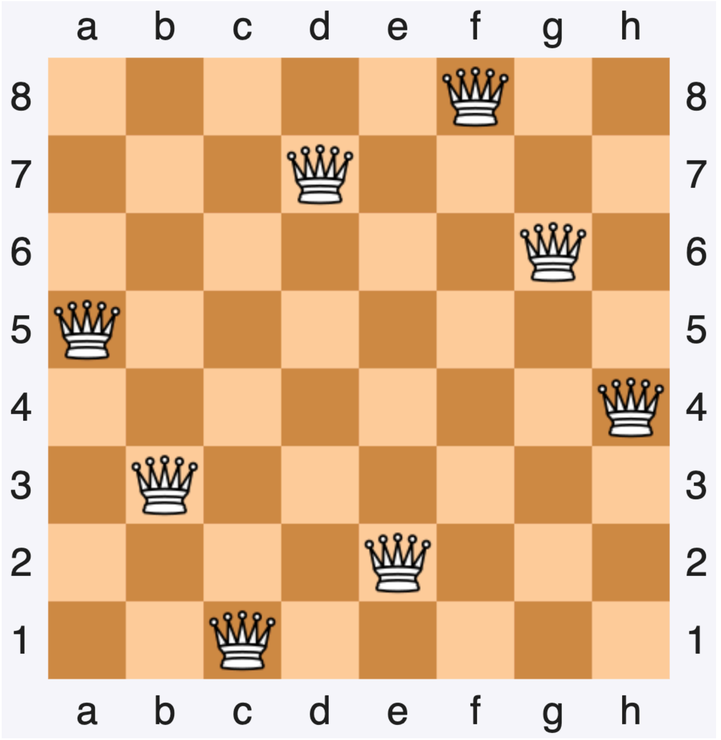
Le tableau vu pour les indices entiers des valeurs possibles

Tableau contenant les valeurs possibles d'une dimension

Les N reines

Voir série

- Sur un échiquier de taille N x N vous devez placer N reines
- Aucune reine ne doit attaquer une autre
- C’est possible!
- Chaque reine est sur une ligne différente (et une colonne différente)
- Il faut trouver les colonnes telles les reines ne s’attaquent pas diagonalement
- A vous de l’implémenter pendant la série!



Qu’est-ce qu’on n’a pas vu?

- Autres fonctions sur les strings

  `<string.h>` — `sscanf`, `strstr`, `strtok`, `atoi`, `etc.`

- Accès au niveau des bits
  - Membres par bit des struct
  - Opérateurs par bit, drapeaux/flags, …
- Union types
- Comment gérer la concurrence
  - Les threads, processes, etc., multiplexage avec `select`

Qu’est-ce qui manque en C?

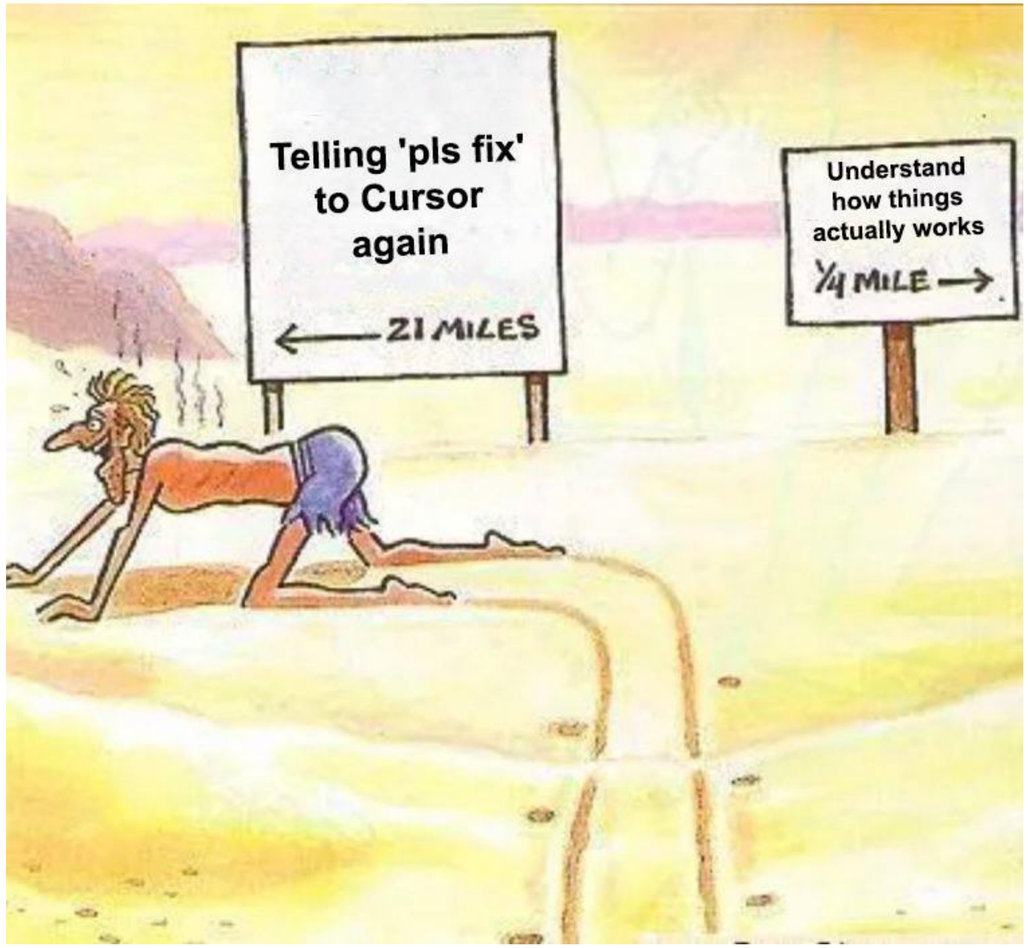
- Les objets
- Les dictionnaires
  - “Key-value stores” — permettent de stocker des clés et des valeurs
  - On ne peut pas écrire des choses comme `age["bob"] = 42`
  - Le “module” `<search.h>` fait ça de manière très primitive
- Une meilleure gestion des chaînes de caractères
- Gestion des erreurs — soit c’est fatal, soit ça peut passer inaperçu…

Quels autres langages faudrait-il connaitre?

- **Python**
  - Versatile, facile à utiliser
  - Interopérabilité avec d'autres langages (C, Java, …)
- **Java** - très utilisé en industrie, microservices, etc.
- **Scala** - Inventé à l'EPFL
  - Langage fonctionnel - compatible avec Java
- **Go**, **Zig**, **Rust**, …

AI for coding

- C’est bien quand on sait ce qu’on veut faire et on veut le faire plus vite
- Ou alors pour coder rapidement qqch qui marche assez bien pour une démo
- Pour un vrai projet il faut comprendre en profondeur le code généré, souvent l’IA hallucine…
- Plus le projet est complexe, moins ça marche
- **A éviter quand on apprend à coder**, c’est comme regarder les solutions directement




Windows
An error has occurred. To continue:
Press Enter to return to Windows, or
Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.
Error: 0E : 016F : BFF9B3D4
Press any key to continue _