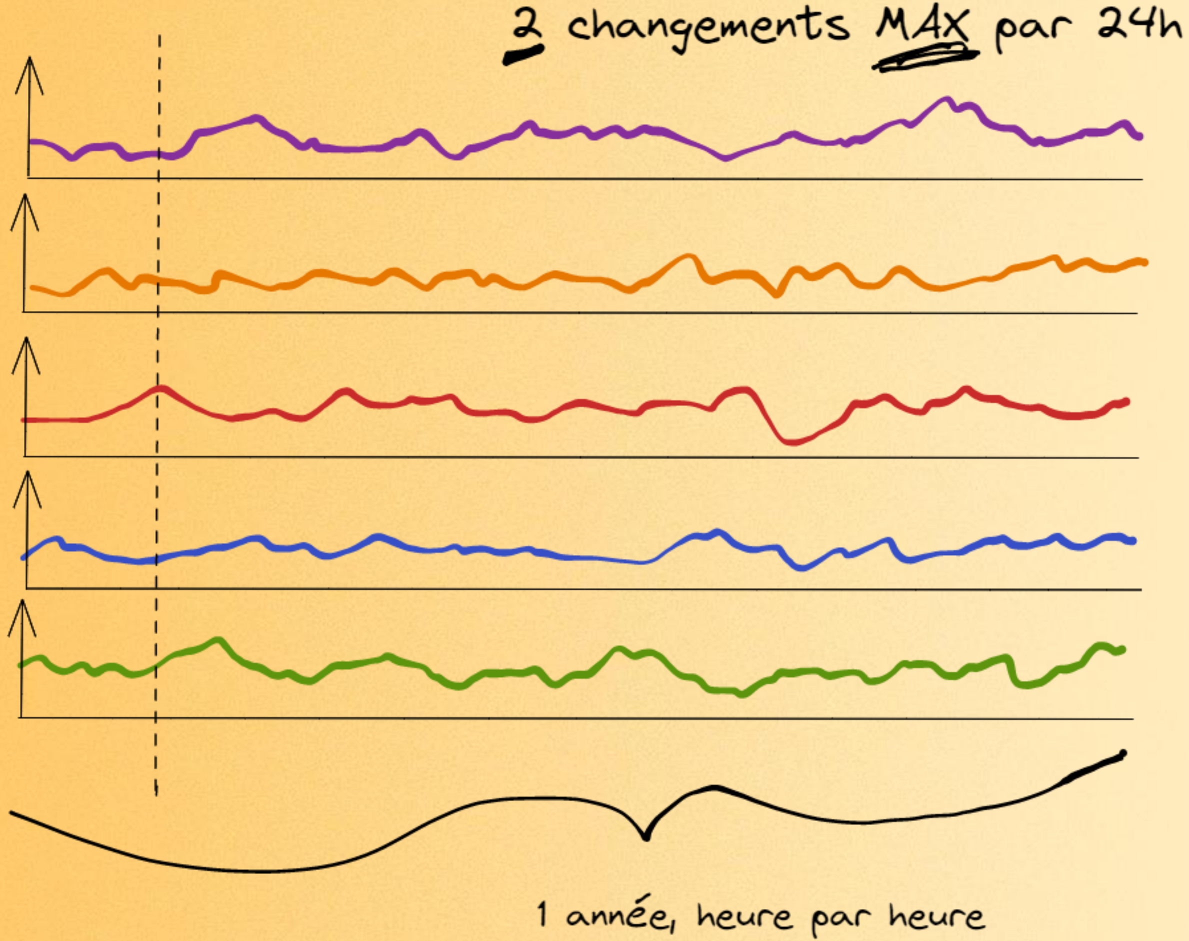
La programmation dynamique pour *sauver* Jules Verne

Gaëtan Eleouet

meritis

Quel est le coût minimum ?

?



Gaëtan Eleouet

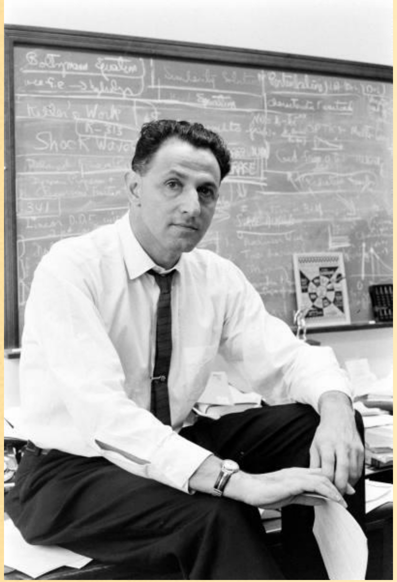
Développeur Java - meritis
@egaetan

1. Une histoire
2. De la Terre à la Lune
3. 20 000 Lieux sous les mers
4. L'Exposition

Une Histoire

La programmation dynamique

Richard Bellman



Crédit :Alfred Eisenstaedt / Time Inc ; C. Imbert.

- Equation de Bellman

$$V(x) = \max_{a \in \Gamma(x)} \{F(x,a) + \beta V(T(x,a))\}$$

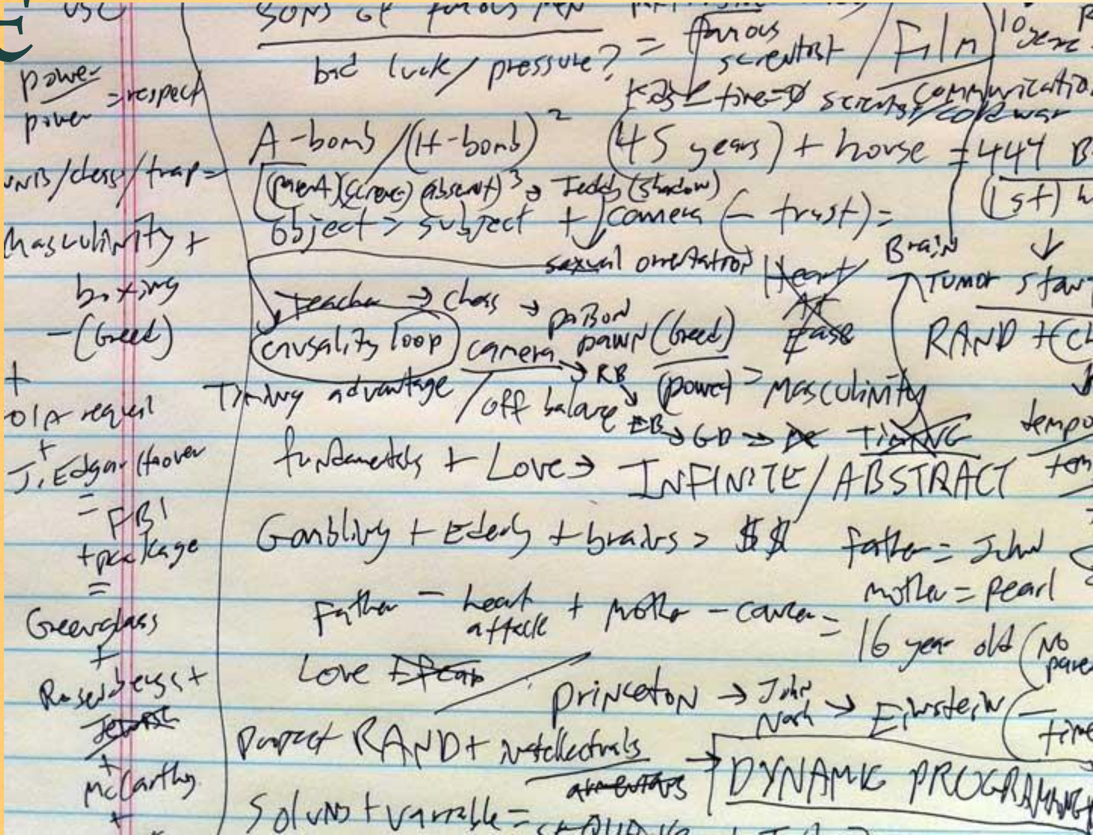
- Equation d'Hamilton–Jacobi–Bellman

$$V_T(x(0),0) = \min_u \left\{ \int_0^T C[x(t),u(t)]dt + D[x(T)] \right\}$$

- « fléau de la dimension » (Curse of dimensionality)

La Programmation dynamique

Histoire



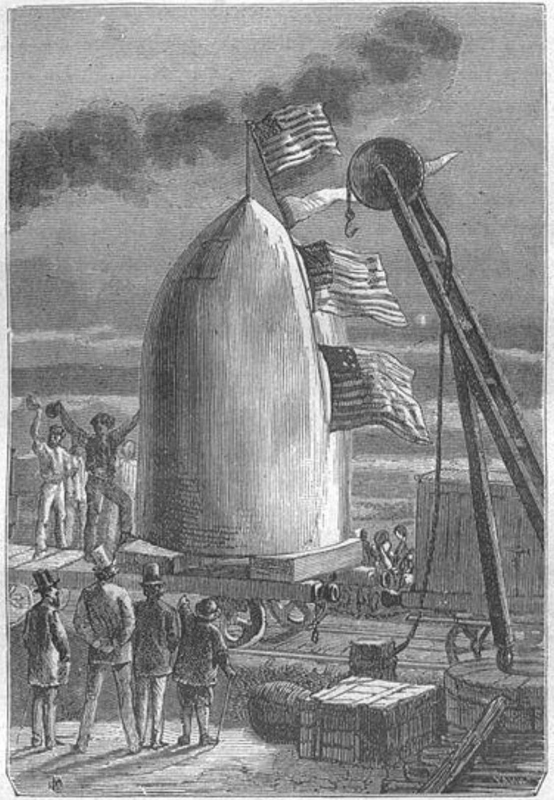
Méthode de création d'algorithme

Pour résoudre des problèmes
d'optimisation ou d'énumération

De la Terre à la Lune

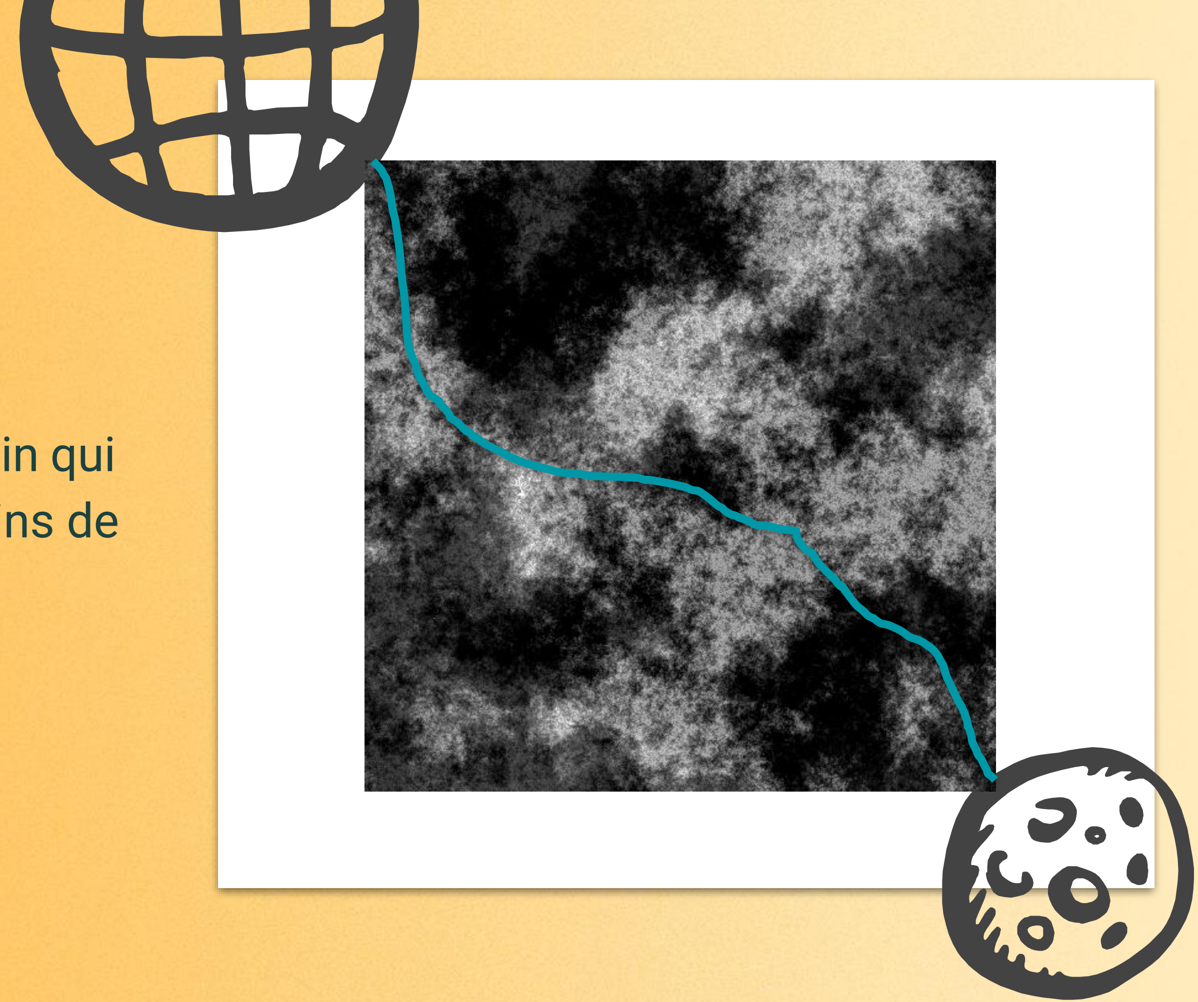
DEVFEST
NANTES 2022

Viser la Lune



L'arrivée du projectile à Stone's-Hill (p. 139).

Trouver le chemin qui rencontre le moins de météorites



Chaque case représente une « densité » de météorites

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

On avance soit :

- vers la droite
- vers le bas

Exemple

100 + 80 + 10 + 20 + 30 + 80 + 95

415

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

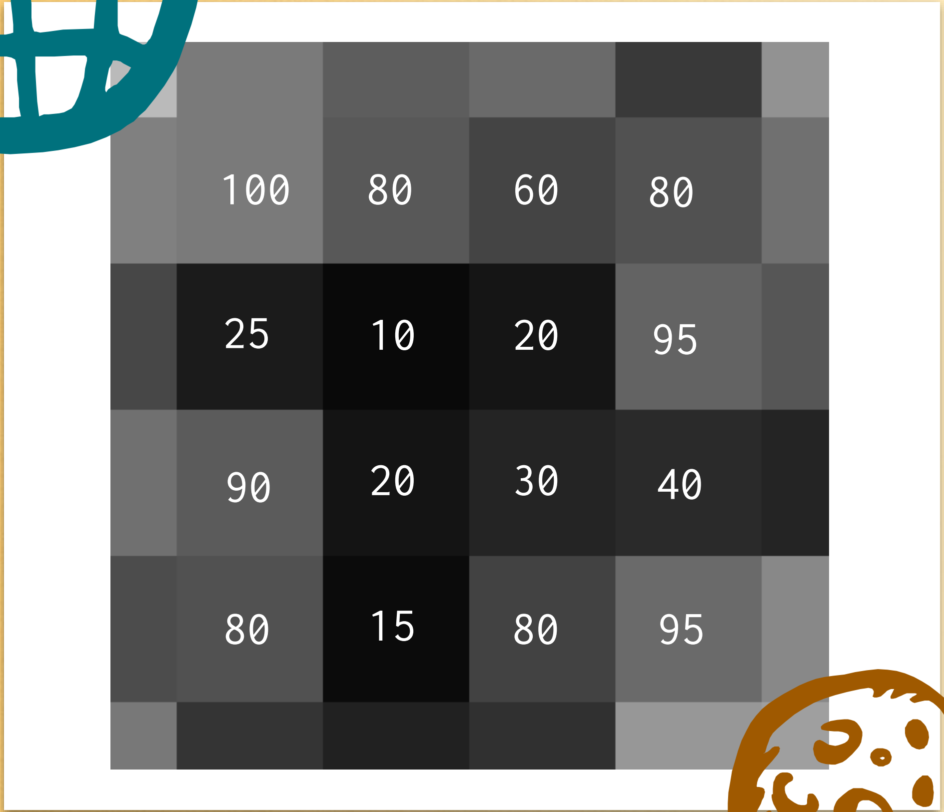
Exemple

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

100 + 25 + 10 + 20 + 15 + 80 + 95

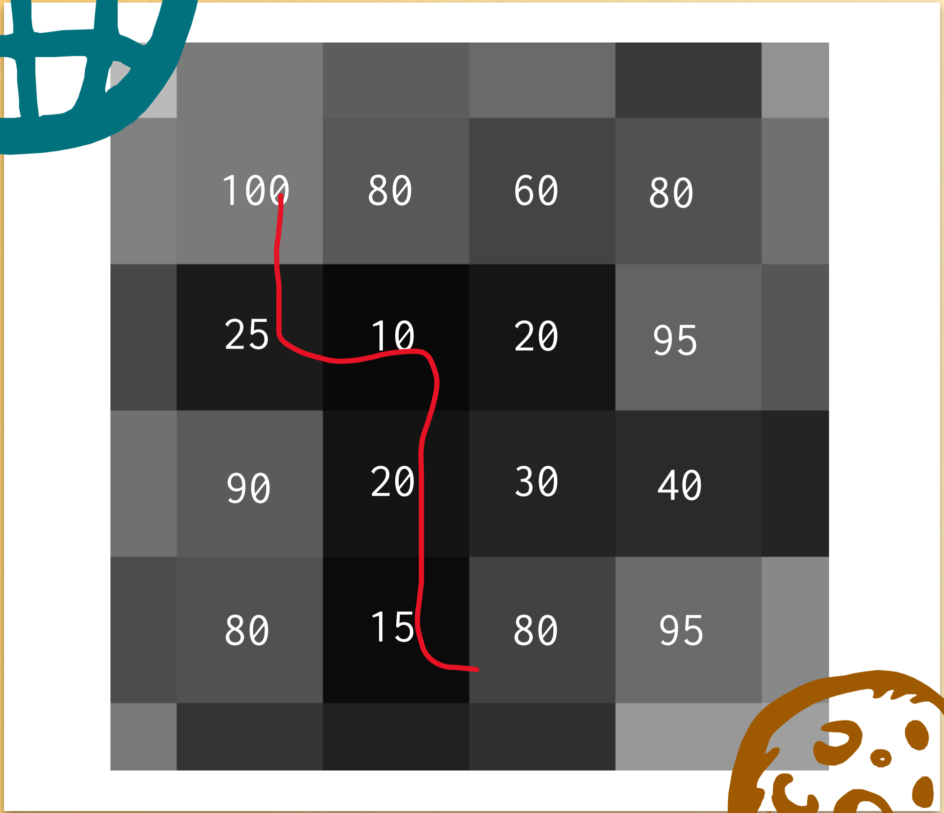
345

A vous !



DEVFEST
NANTES 2022

Le minimum à chaque étape n’est pas la minimum global

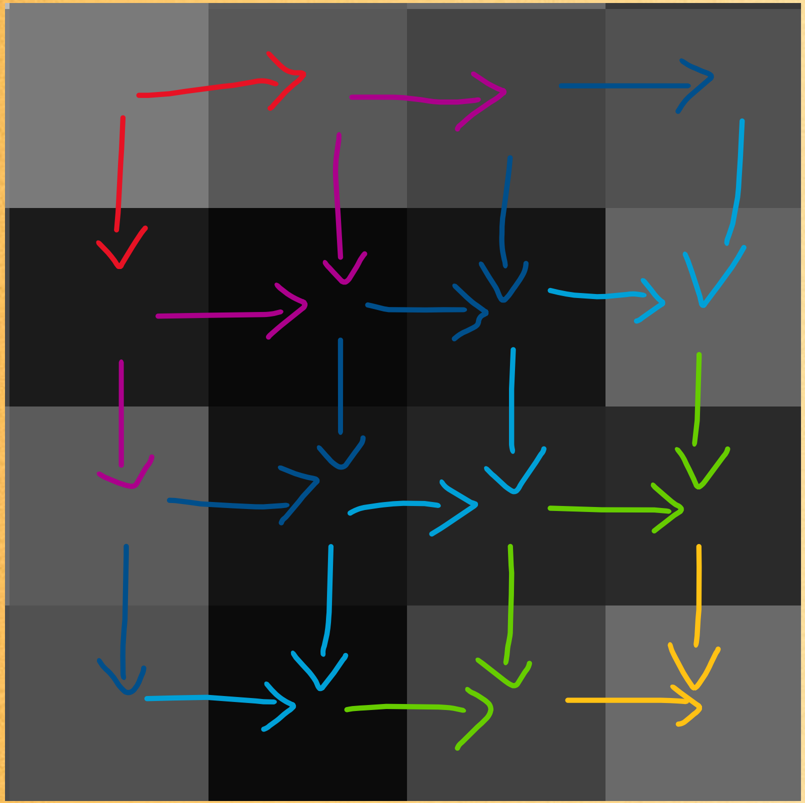


Recherche exhaustive (Force Brute)

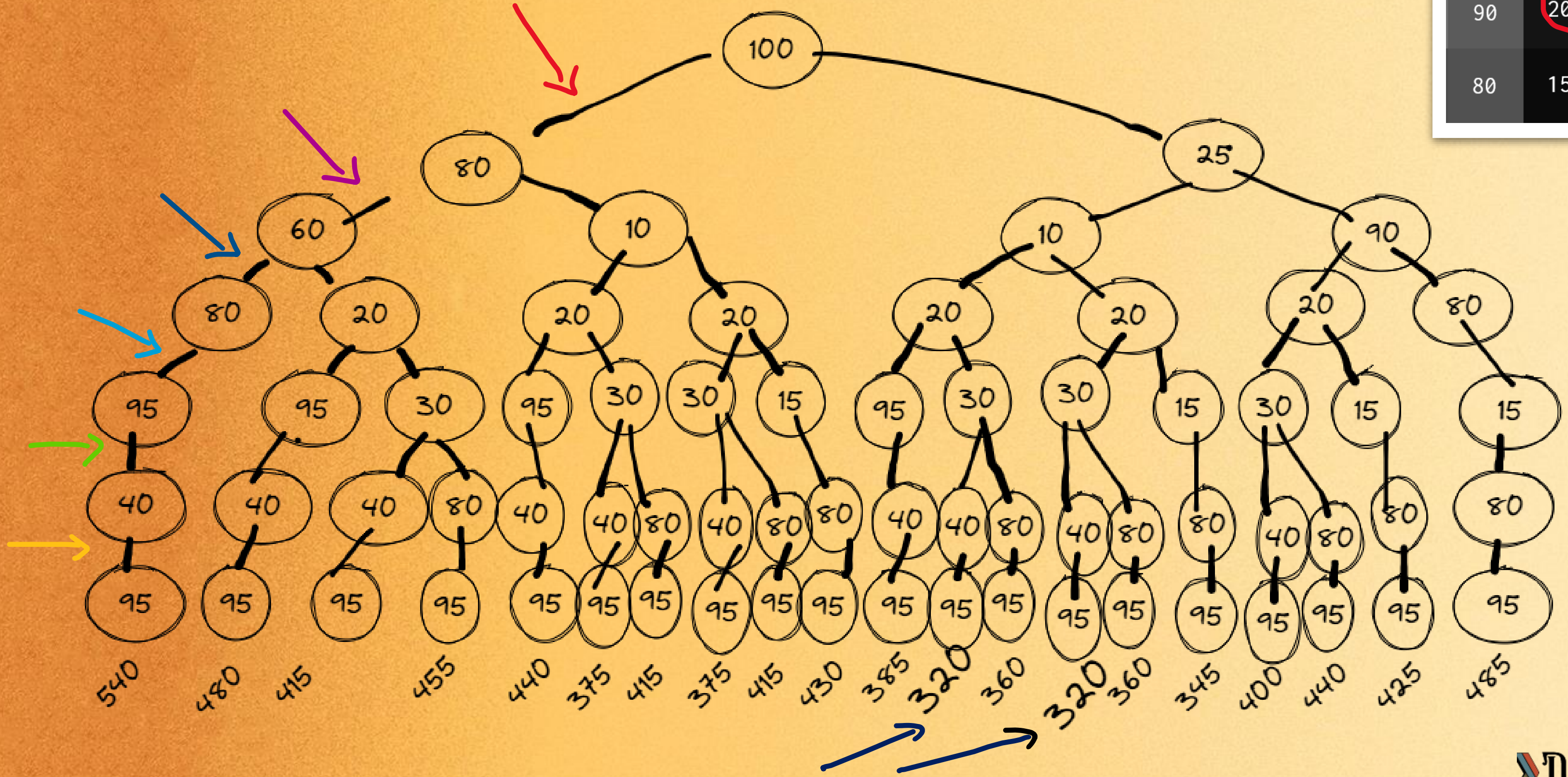
Tester tous le chemins possibles

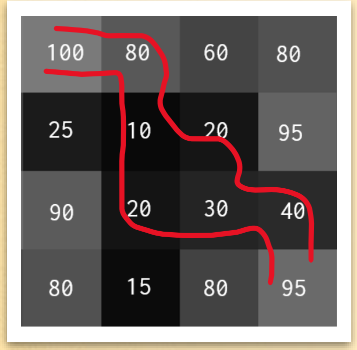
Retenir le **meilleur**

Tous les chemins



Tous les chemins (4x4)





L'ALGORITHME

```
Calculer (position) {

        Si position est destination

                alors cout(position)

        Sinon si position est sur un bord

                alors cout(position) + Calculer(position suivante)

        Sinon

                cout(position)  + minimum(Calculer(position bas),
                                          Calculer(position droite))

}
```

Le code

```
int compute(Grid grid, Position p) {
    if (p.x < grid.limit) {
        if (p.y < grid.limit) {
            return grid.at(p)
                + min(compute(grid, new Position(p.x + 1, p.y)),
                      compute(grid, new Position(p.x, p.y + 1)));
        }
        else {
            return grid.at(p) + compute(grid, new Position(p.x + 1, p.y));
        }
    }
    else if (p.y < grid.limit) {
        return grid.at(p) + compute(grid, new Position(p.x, p.y + 1));
    }
    else {
        return grid.at(p);
    }
}
```

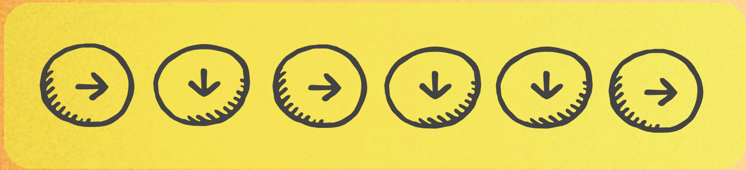
Le code

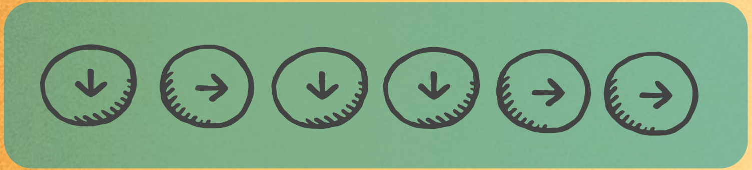
```
int compute(Grid grid, Position p) {
    if (p.x < grid.limit) {
        if (p.y < grid.limit) {
            return grid.at(p)
                + min(compute(grid, new Position(p.x + 1, p.y)),
                      compute(grid, new Position(p.x, p.y + 1)));
        }
        else {
            return grid.at(p) + compute(grid, new Position(p.x + 1, p.y));
        }
    }
    else if (p.y < grid.limit) {
        return grid.at(p) + compute(grid, new Position(p.x, p.y + 1));
    }
    else {
        return grid.at(p);
    }
}
```

Sur un bord

Destination

Complexité ?

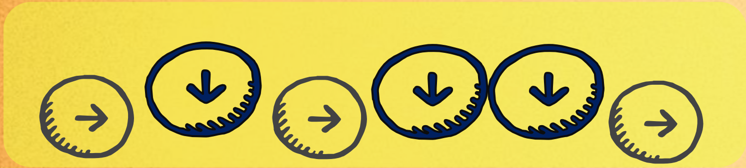


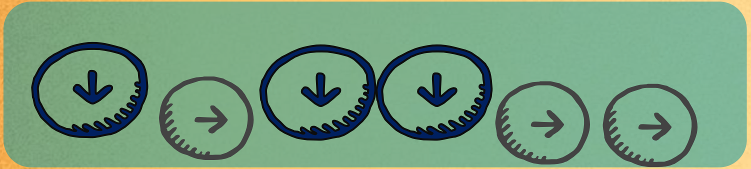


100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

Complexité ?





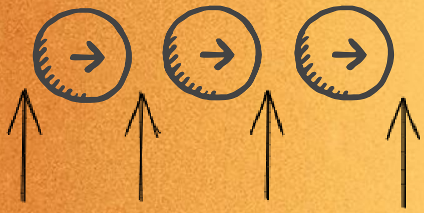
100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

DEVFEST
NANTES 2022

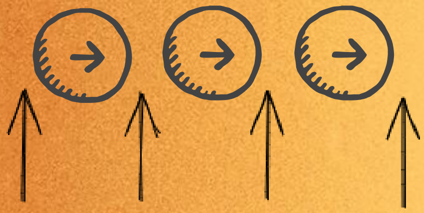
Complexité

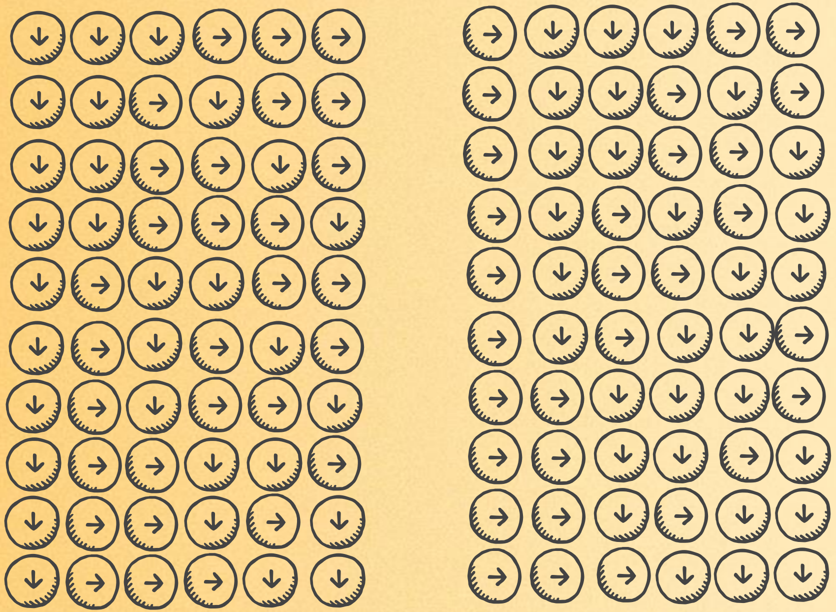
Cela revient à choisir où l'on place
les 3 ↓ , parmi les 3 → *(4 choix)*



Complexité

Cela revient à choisir où l'on place les 3 ↓ , parmi les 3 →





Complexité

Combinaisons avec répétitions, choisir k parmi n :

$$D_n^k = C_{n+k-1}^k = \frac{(n+k-1)!}{k! \cdot (n-1)!}.$$

Avec N = taille,

n = N+1 et k = N

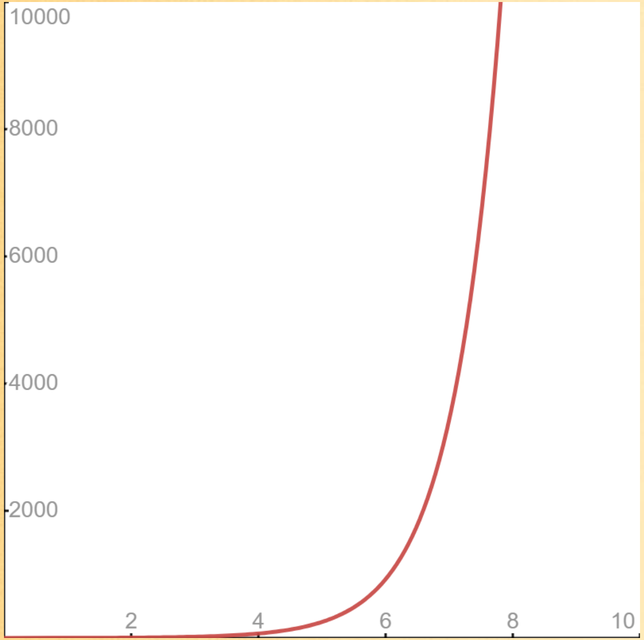
Soit : $$\frac{(2 * N)!}{(N!)^2}$$

Oooops

Pour n=512

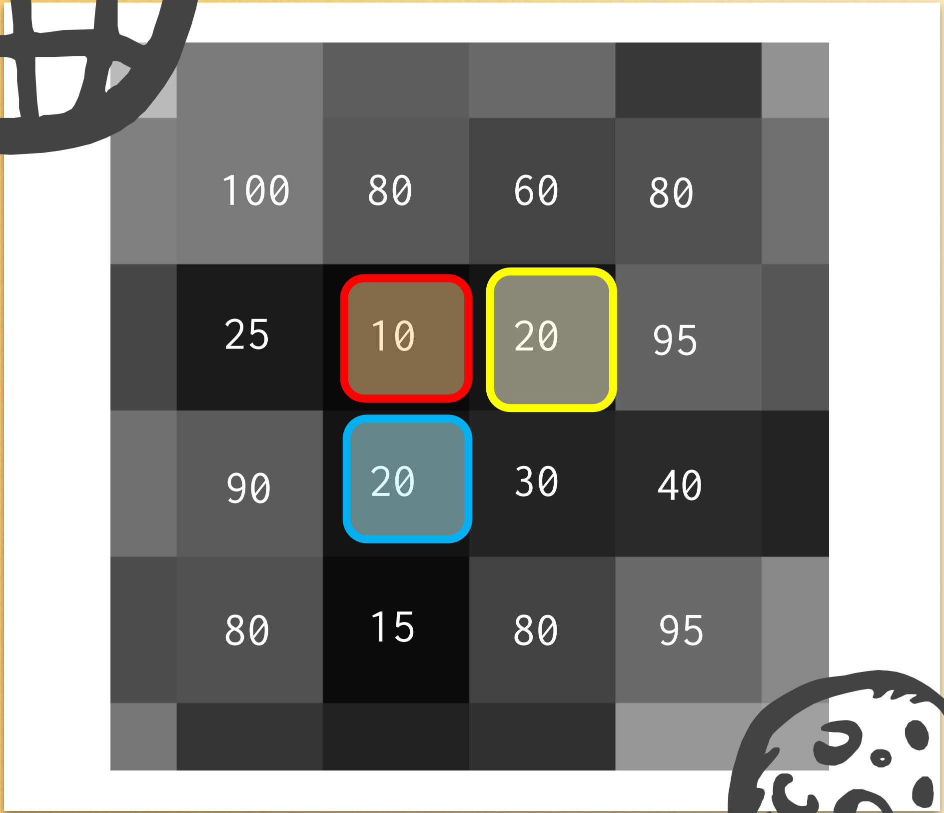
$$\frac{(2 * N)!}{(N!)^2}$$

448125455209897081002416485048133318001530785
906773699441608789940477370661143964479108414
007291406034616943401861860280300750167237649
685869987398362661606247167585150557210202515
933540109055902782852210522976011490037704775
010193851160493255364746251743844451364876533
269450028332840221386876395657391 3670



Pour calculer

On utilise et

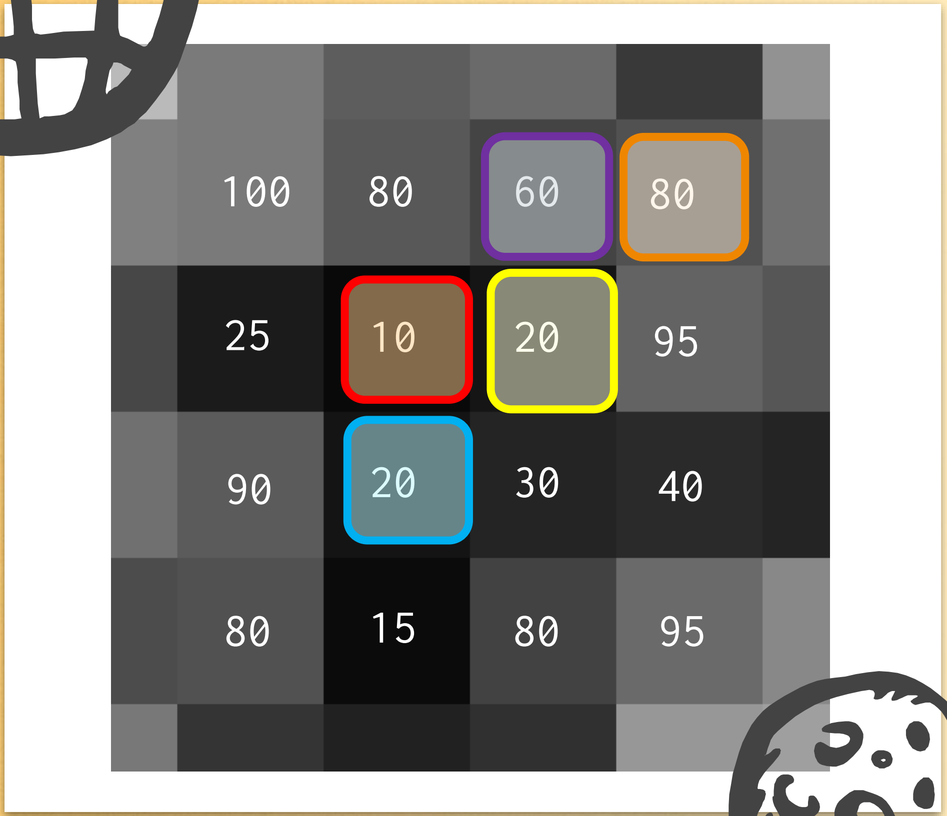


Pour calculer

On utilise et

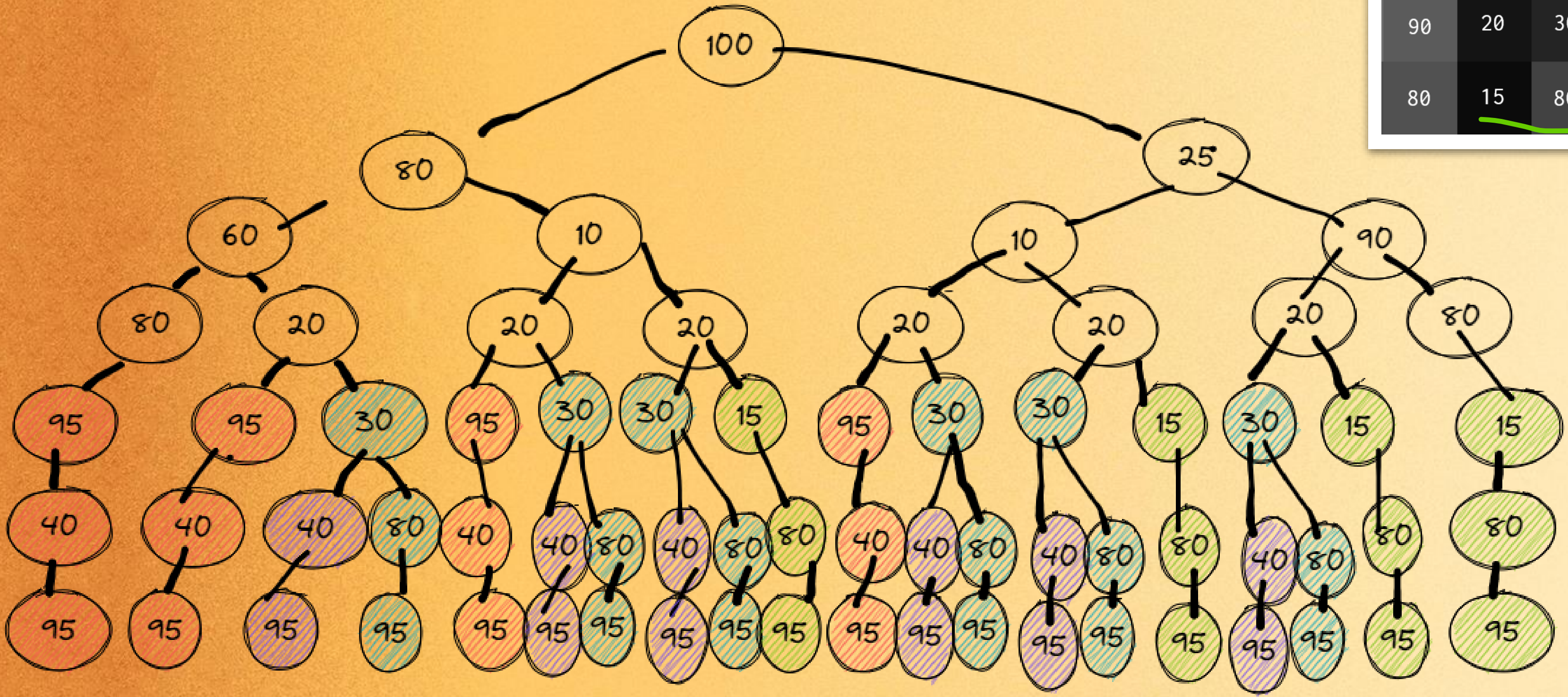
Pour calculer

On utilise et



DEVFEST NANTES 2022

Beaucoup de répétitions



100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95

Récursion avec **mémoïsation**

On ajoute un **cache** à notre récursion

L'ALGO

Calculer (position) {

Si position est dans le cache, alors utiliser cette valeur

Sinon calculer et mettre dans le cache

Si position *est* destination

alors cout(position)

Sinon si position *est sur un bord*

alors cout(position) + **Calculer**(position *suivante*)

Sinon

cout(position) + minimum(**Calculer**(position bas),
Calculer(position droite))

}

DEVFEST NANTES 2022

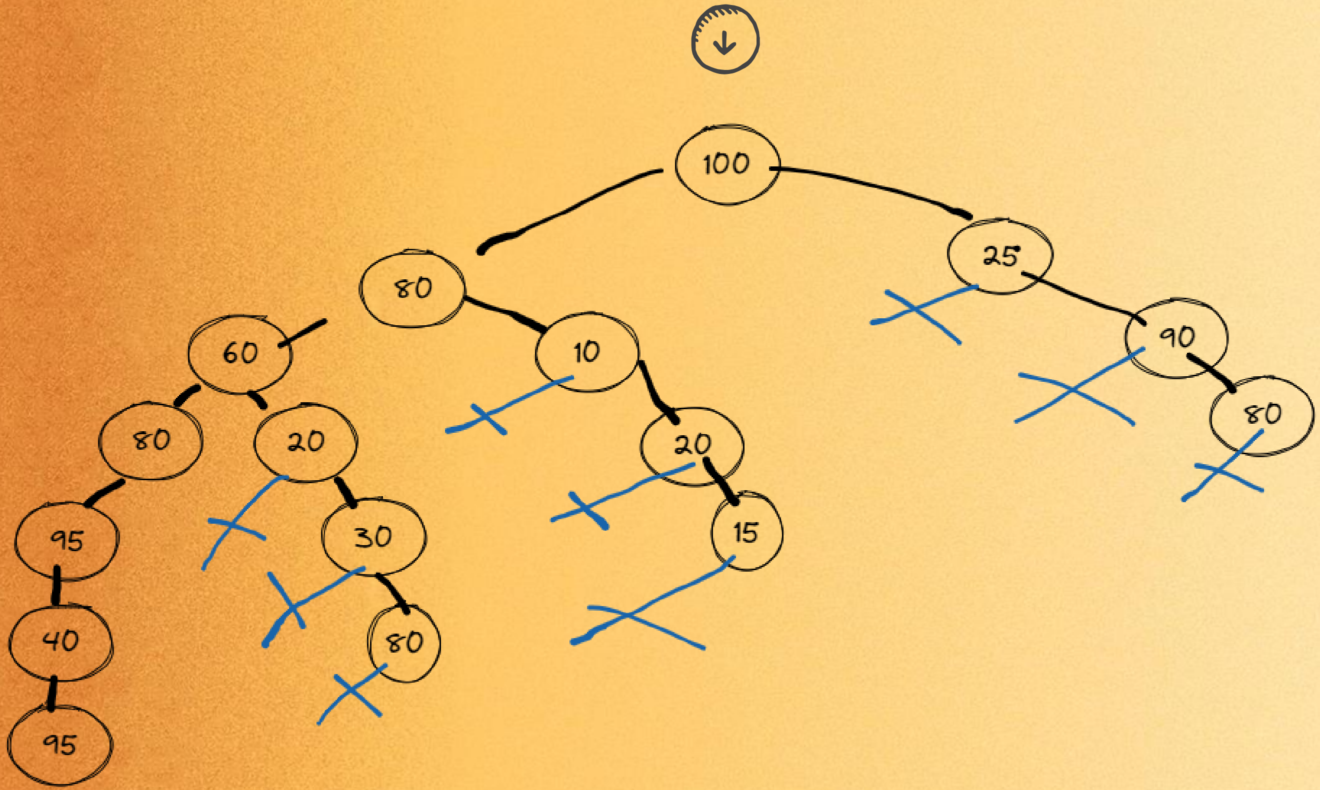
Sous-Problèmes qui se chevauchent (overlapping)

Mémoriser ce qui est calculé !

```java
Cache<Position> cache = new Cache<>();
int computeMemo(Grid grid, Position p) {
    if (cache.doesntContains(p)) {
        if (p.x < grid.limit) {
            if (p.y < grid.limit) {
                cache.memo(p, grid.at(p)
                   + min(computeMemo(grid, new Position(p.x + 1, p.y)),
                         computeMemo(grid, new Position(p.x, p.y + 1))));
            }
            else {
                cache.memo(p, grid.at(p) + computeMemo(grid, new Position(p.x + 1, p.y)));
            }
        }
        else if (p.y < grid.limit) {
            cache.memo(p, grid.at(p) + computeMemo(grid, new Position(p.x, p.y + 1)));
        }
        else {
            cache.memo(p, grid.at(p));
        }
    }
    return cache.get(p);
}
```

Une seule fois par Case

100	80	60	80
25	10	20	95
90	20	30	40
80	15	80	95



Complexité

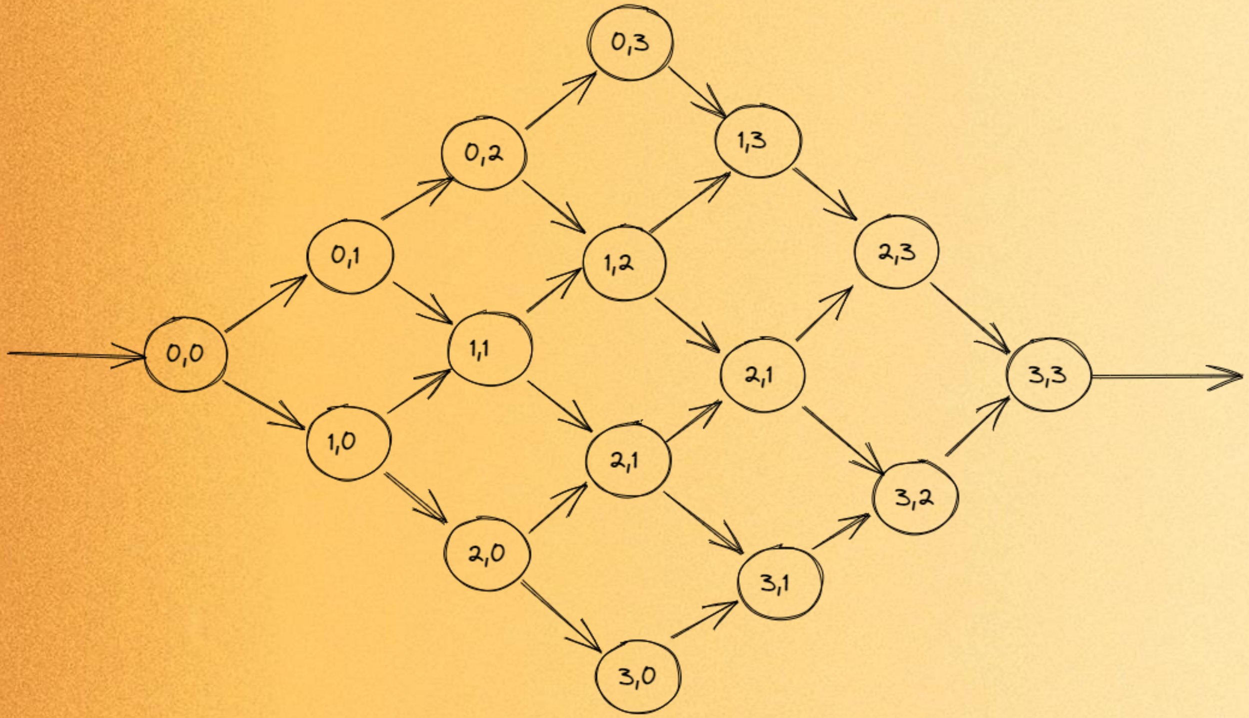
On calcule une seule fois chaque case de la grille

Avec N = taille,

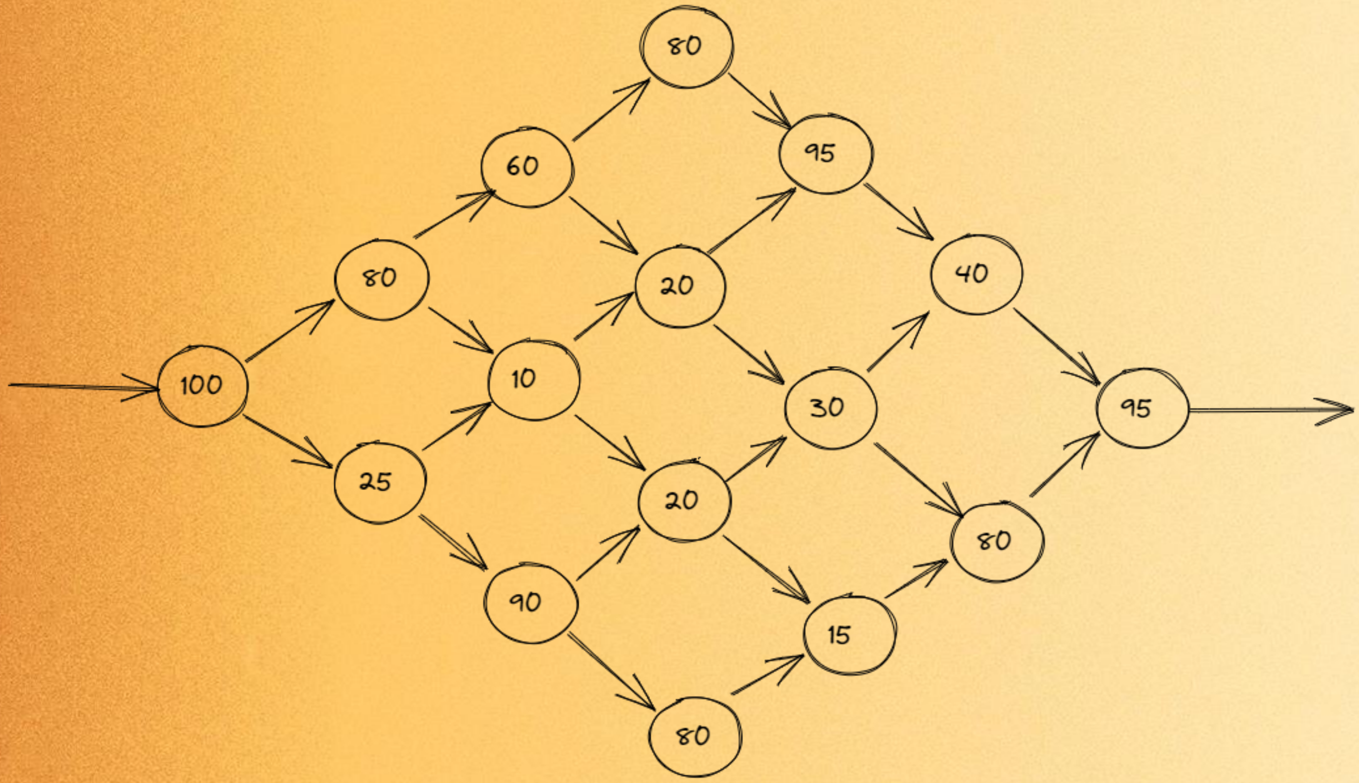
Soit : N^2

(en temps et en espace)

Parcours de Graph

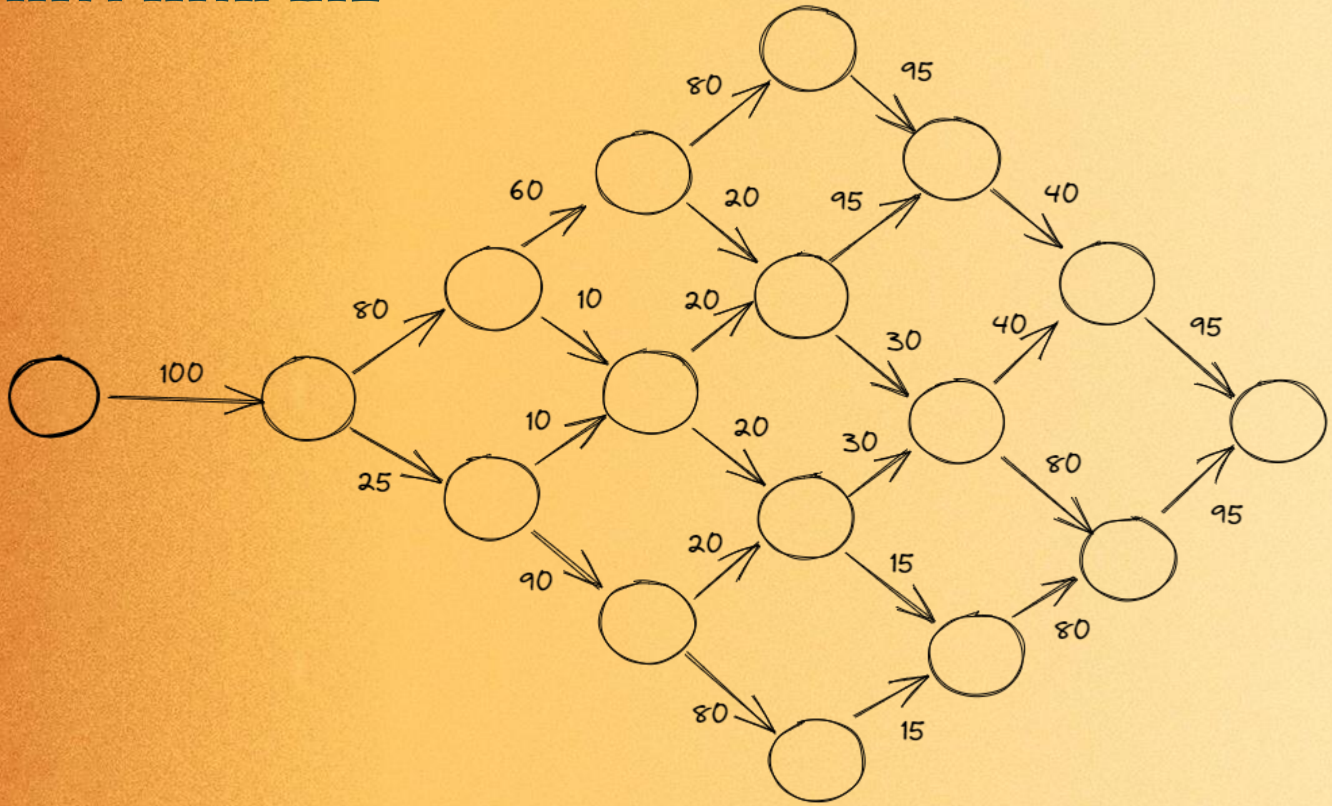


Chemin minimal



DEVFEST NANTES 2022

Chemin minimal



Expression

Chaque problème de programmation dynamique peut être formulé comme un **parcours** de graph

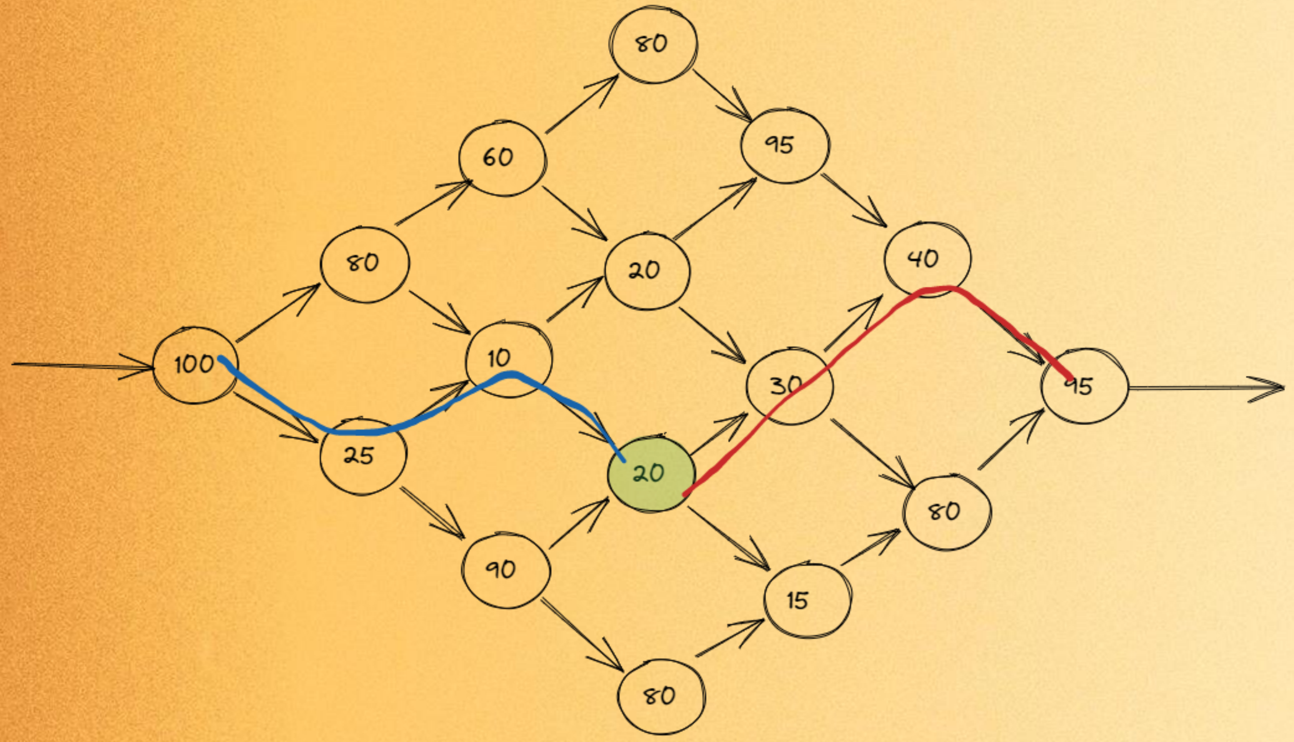
Chaque « nœud » représente un « état » et le problème consiste en une succession de décision de transition entre ces « états »

Principe d'optimalité de Bellman

Un chemin optimal est formé de sous-chemins optimaux

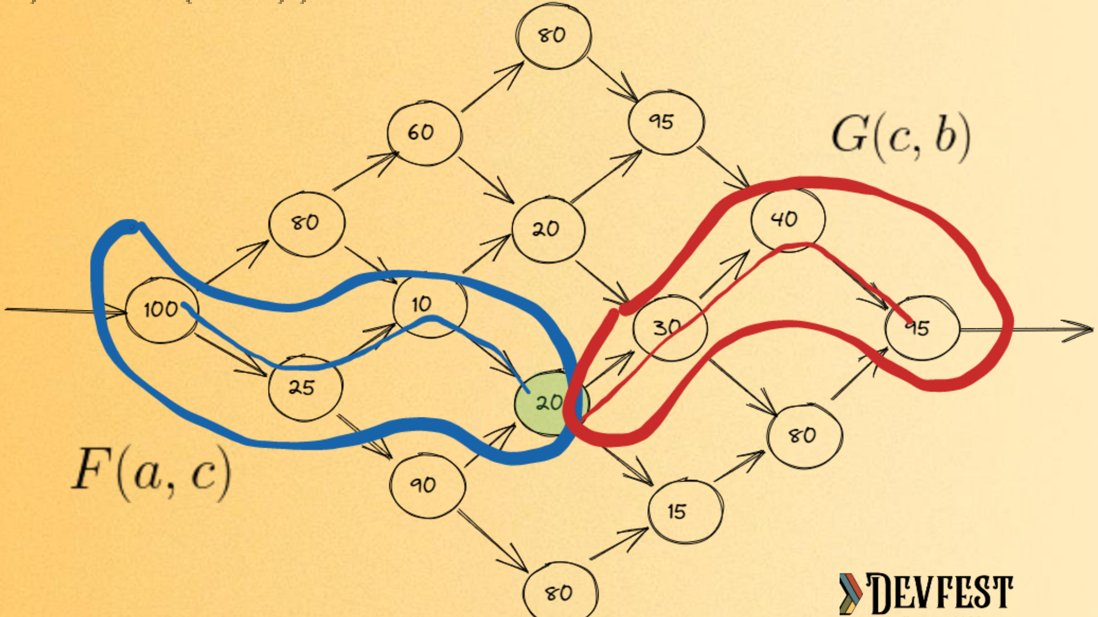
Si $\mathfrak{C}$ est un chemin optimal entre A et B et si C appartient à $\mathfrak{C}$, alors les sous-chemins de A à C et de C à B dans $\mathfrak{C}$ sont optimaux

Chemin minimal



Principe D'optimalité de Bellman

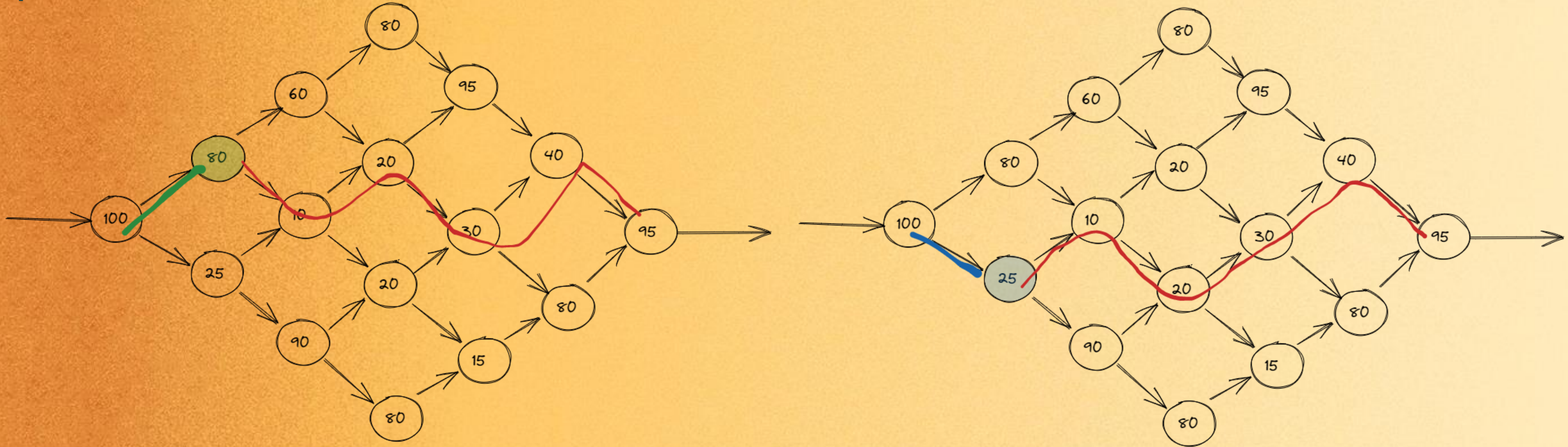
$$F(a,b) = min_c(F(a,c) + G(c,b))$$



DEVFEST NANTES 2022

Enumérer

La programmation dynamique va essayer **toutes** les possibilités

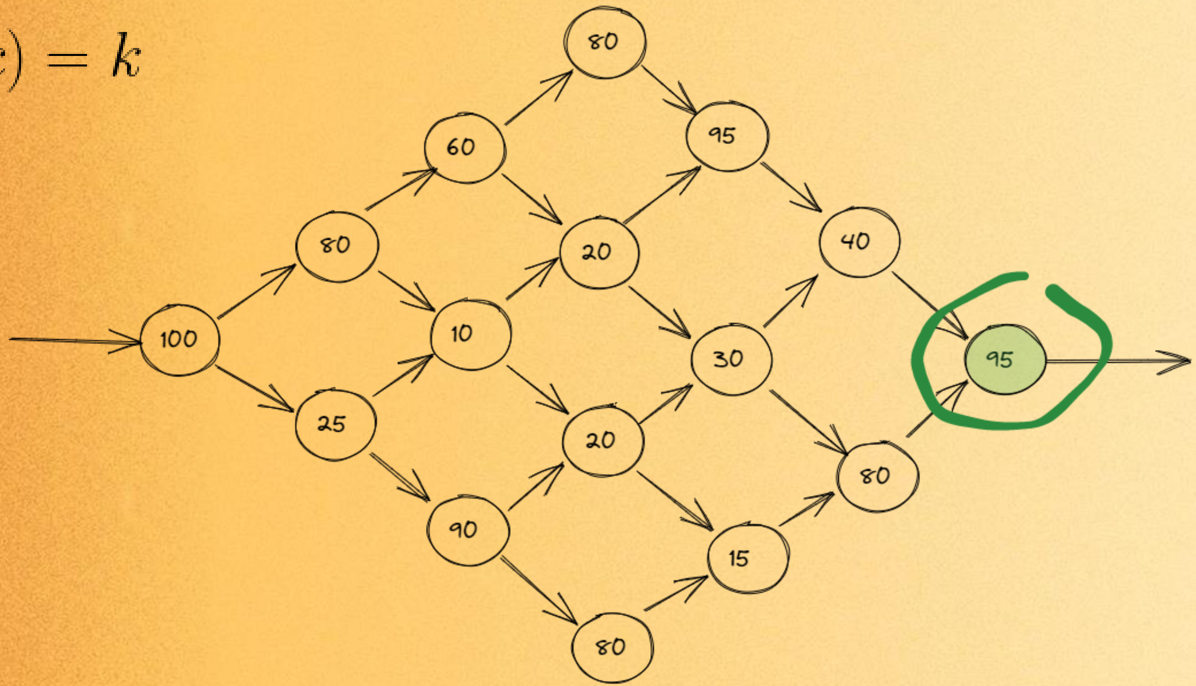


*Et garder la **meilleure***

LA CONDITION D'ARRÊT

Le cas nominal

$$F(c, c) = k$$

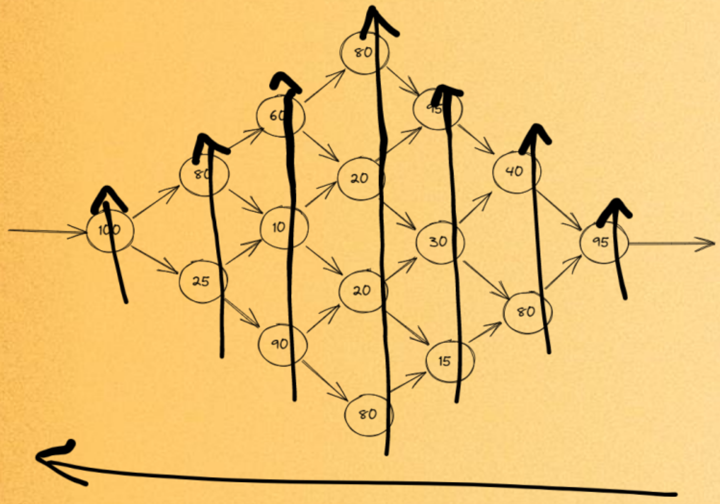


DEVFEST NANTES 2022

L'expression de récursion

$$\begin{cases} F(a,b) = min_c(F(a,c) + G(c,b)) \\ F(b,b) = k \end{cases}$$

Exprime un ordre topologique sur les nœuds du graph acyclique



Version tabulée

```java
int computeTable(Grid grid, Position p) {
    int[][] ligne = new int[grid.limit + 1][grid.limit + 1];
    for (int l = grid.limit; l >= 0; l--) {
        for (int c = grid.limit; c >= 0; c--) {
            if (l == grid.limit && c == grid.limit) {
                ligne[c][l] = grid.at(new Position(l, c));
            } else if (l == grid.limit) {
                ligne[c][l] = grid.at(new Position(l, c)) + ligne[c + 1][l];
            } else if (c == grid.limit) {
                ligne[c][l] = grid.at(new Position(l, c)) + ligne[c][l + 1];
            } else {
                ligne[c][l] = grid.at(new Position(l, c))
                            + min(ligne[c][l+1], ligne[c+1][l]);
            }
        }
    }
    return ligne[0][0];
}
```

On construit le cache

Construction Tabulaire

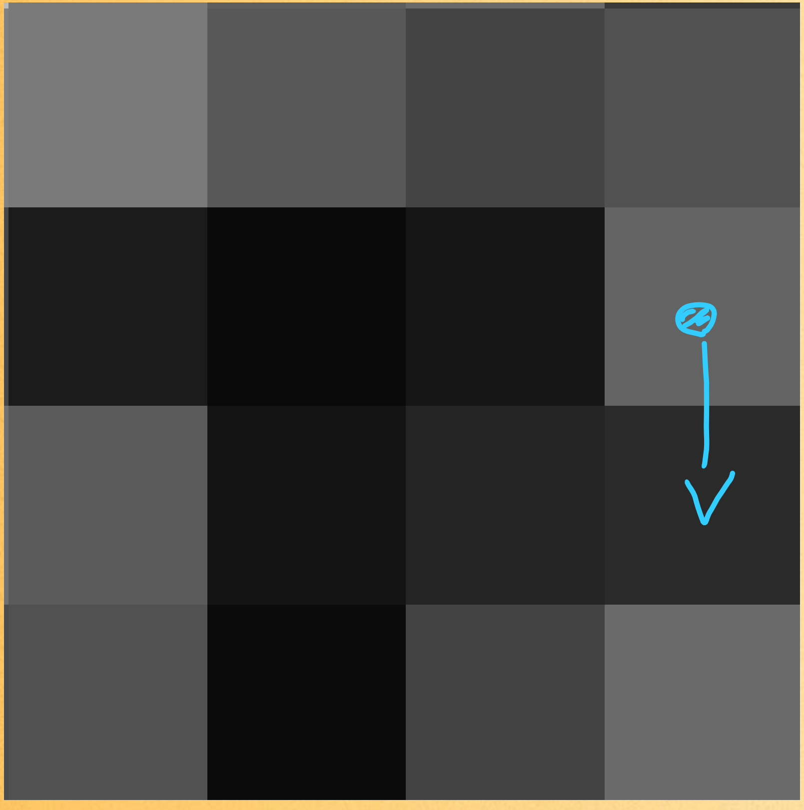
On construit la solution par « remplissage »

Version **itérative** de l'algorithme récursif

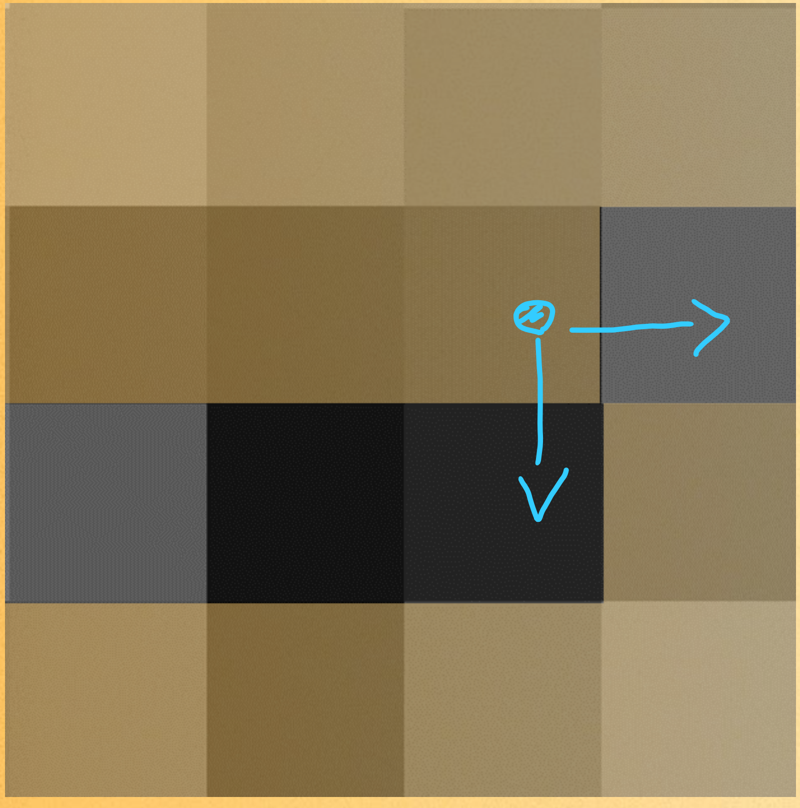
Equivalent à « pré-remplir » le cache

Même **complexité** temporelle

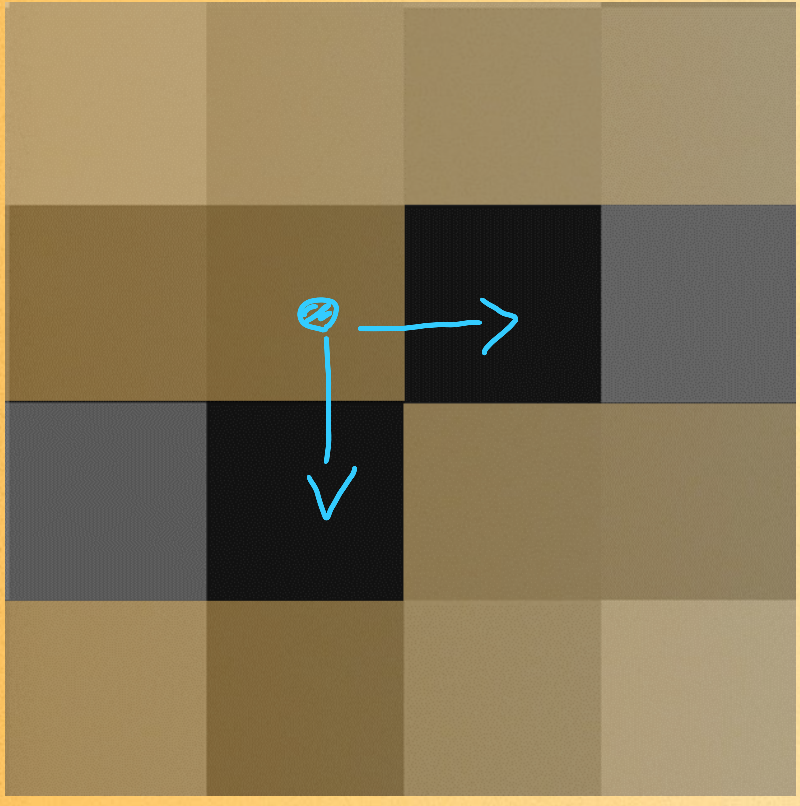
Construction itérative



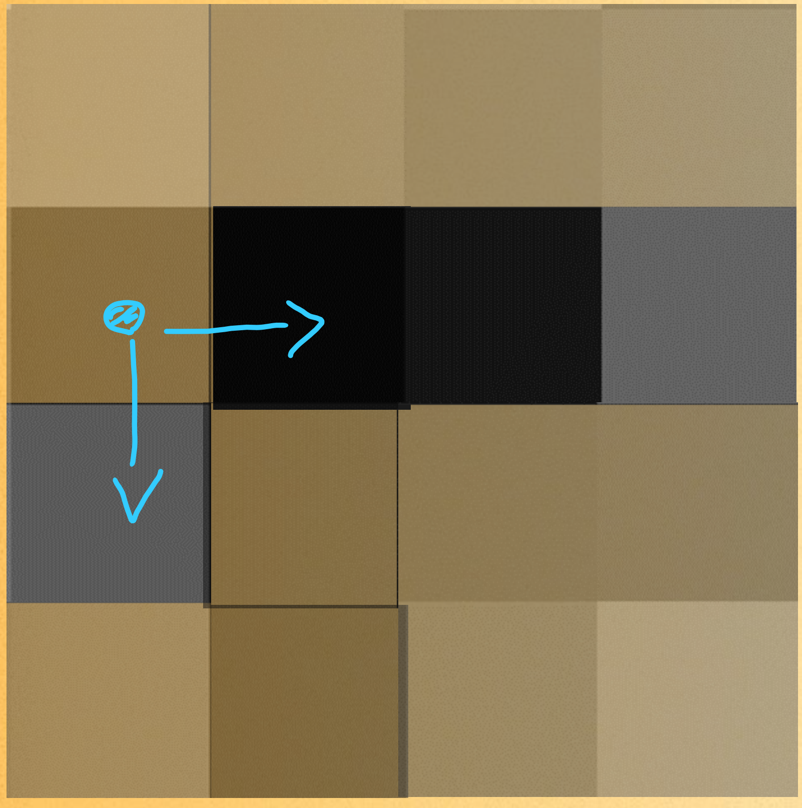
Dépendances des calculs



Dépendances des calculs



Dépendances des calculs



Construction Tabulaire

On peut conserver uniquement les **états** nécessaires

Economie d’espace

Espace réduit

```
int computeLigne(Grid grid, Position p) {
    int[] ligne = new int[grid.limit + 1];
    for (int l = grid.limit; l >= 0; l--) {
        for (int c = grid.limit; c >= 0; c--) {
            if (l == grid.limit && c == grid.limit) {
                ligne[c] = grid.at(new Position(l, c));
            } else if (l == grid.limit) {
                ligne[c] = grid.at(new Position(l, c)) + ligne[c + 1];
            } else if (c == grid.limit) {
                ligne[c] = grid.at(new Position(l, c)) + ligne[c];
            } else {
                ligne[c] = grid.at(new Position(l, c)) + min(ligne[c], ligne[c + 1]);
            }
        }
    }
return ligne[0];
}
```

Une seule ligne

Complexité

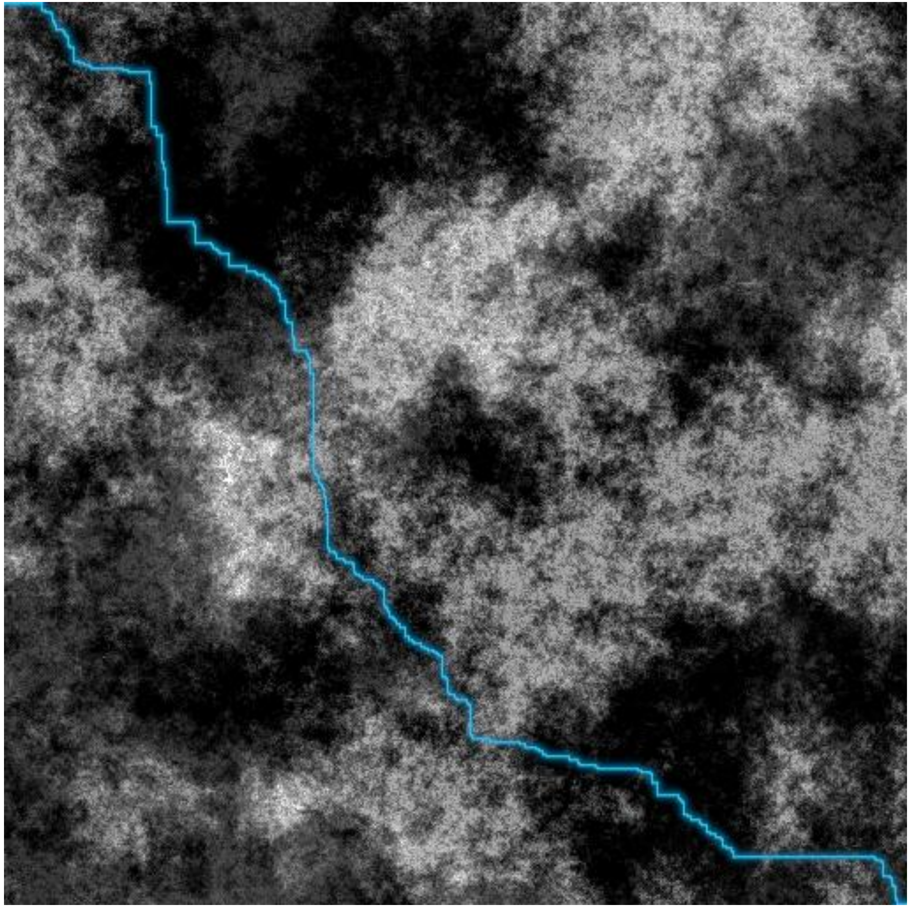
On calcule une seule fois chaque case de la grille

Avec N = taille,

Soit : N^2

(Seulement N en espace ! L'état à chaque itération est de taille N)

Résultat



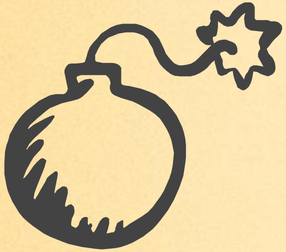
Bottom-Up *vs* Top-Down

Récursion avec **Mémoïsation**

Top-Down

Construction tabulaire **itérative**

Bottom-Up

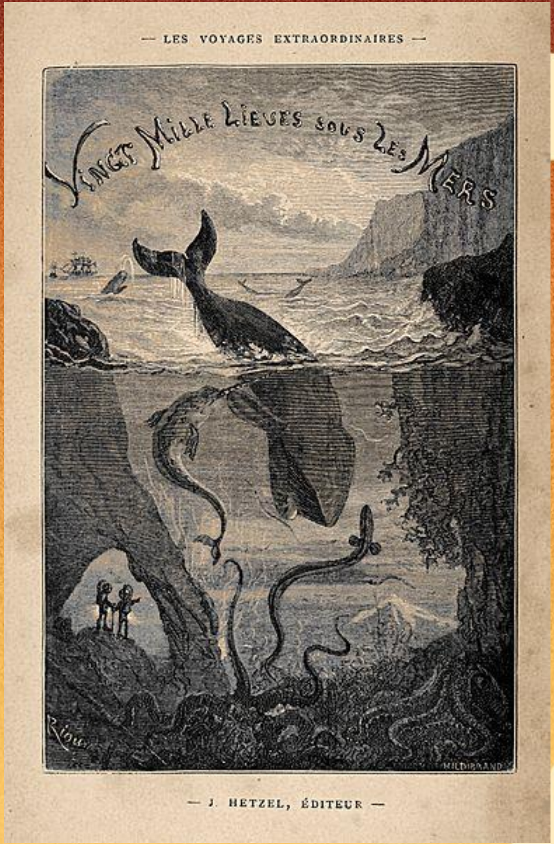


20 000 LIEUX SOUS LES MERS

DEVFEST

NANTES 2022

« Vous êtes venus surprendre un secret que nul homme au monde ne doit pénétrer, le secret de toute mon existence ! Et vous croyez que je vais vous renvoyer sur cette terre qui ne doit plus me connaître ! Jamais ! En vous retenant, ce n'est pas vous que je garde, c'est moi-même ! »



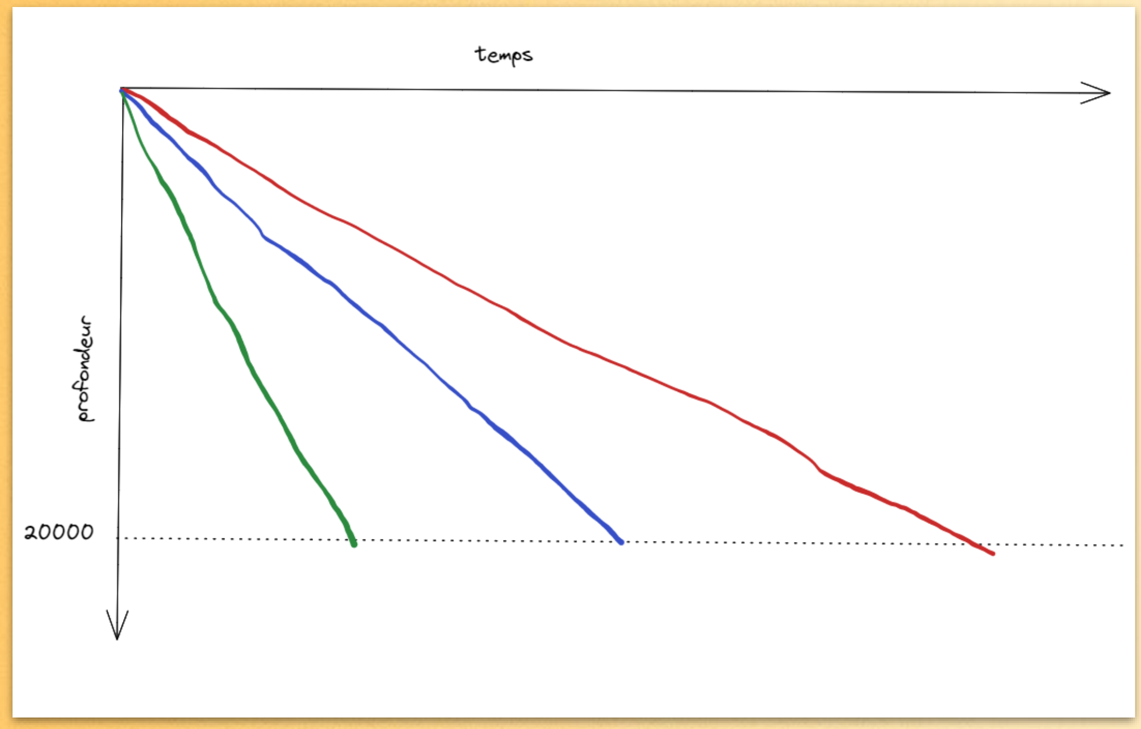
DEVFEST
NANTES 2022

Le Nautilus descend dans les profondeurs

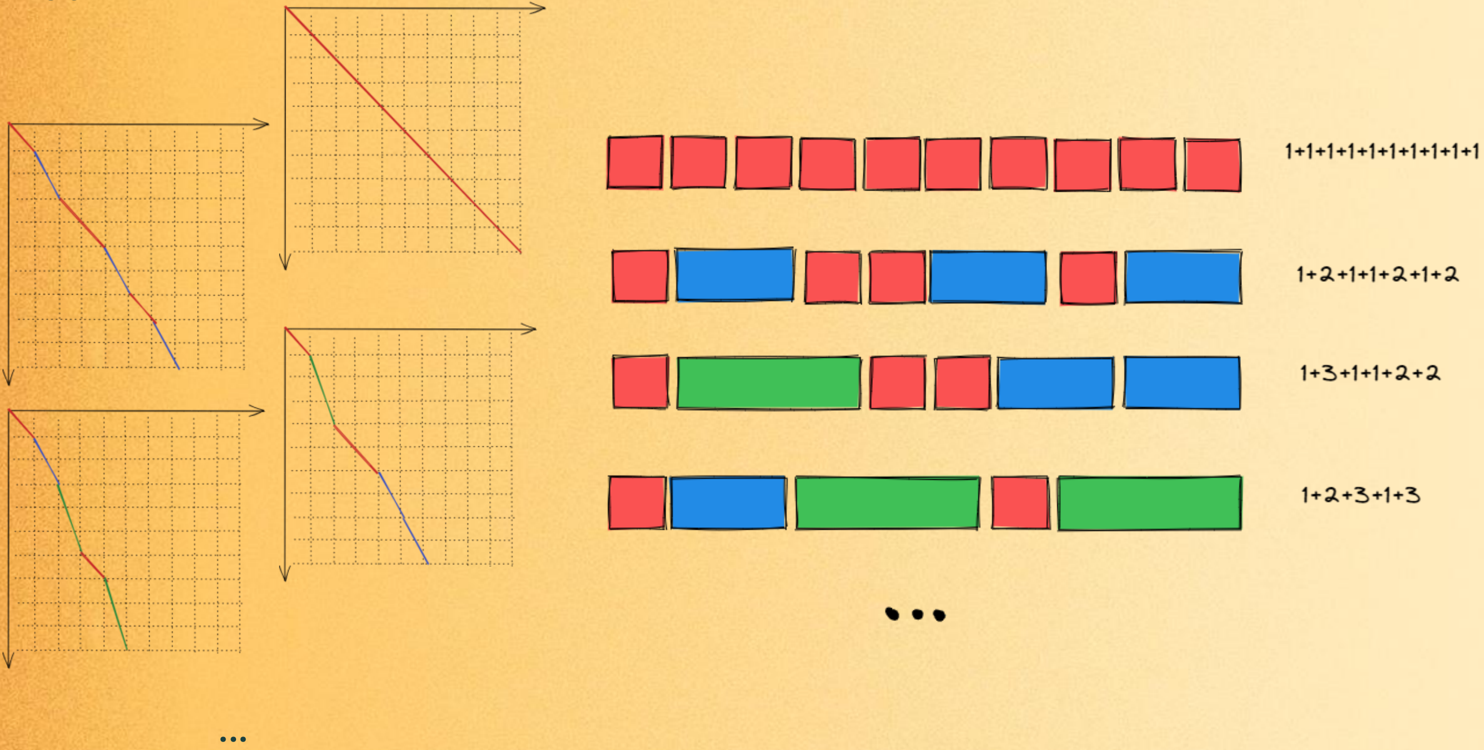
A chaque minute, il descend de 1, 2 ou 3 lieux

De combien de façons différentes peut-il descendre jusqu'à 20 000 lieux ?

De combien de façons différentes peut-il descendre les 20 000 lieux ?



Exemple pour 10 lieux



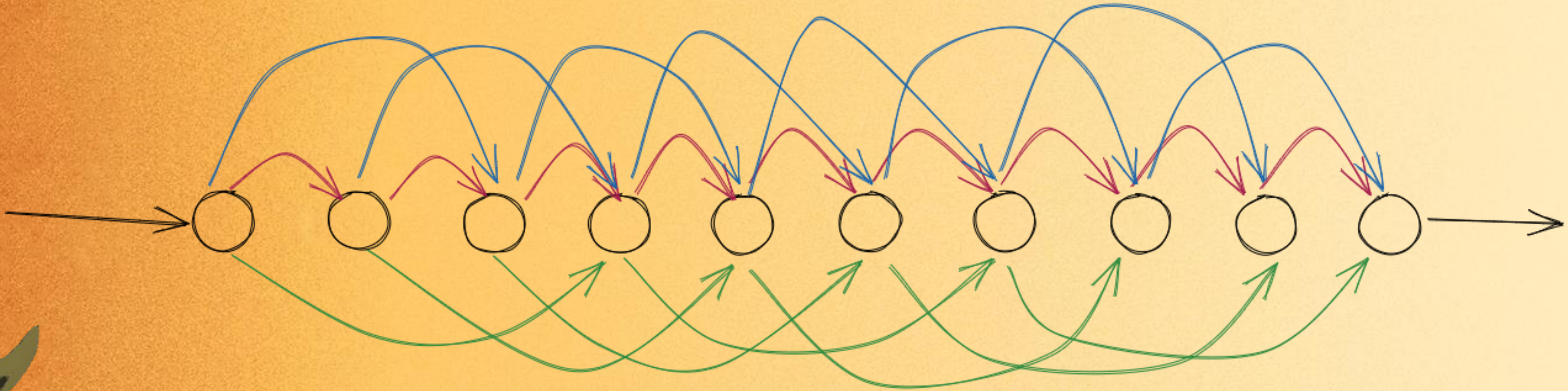
A) 5768

B) 274

C) 81

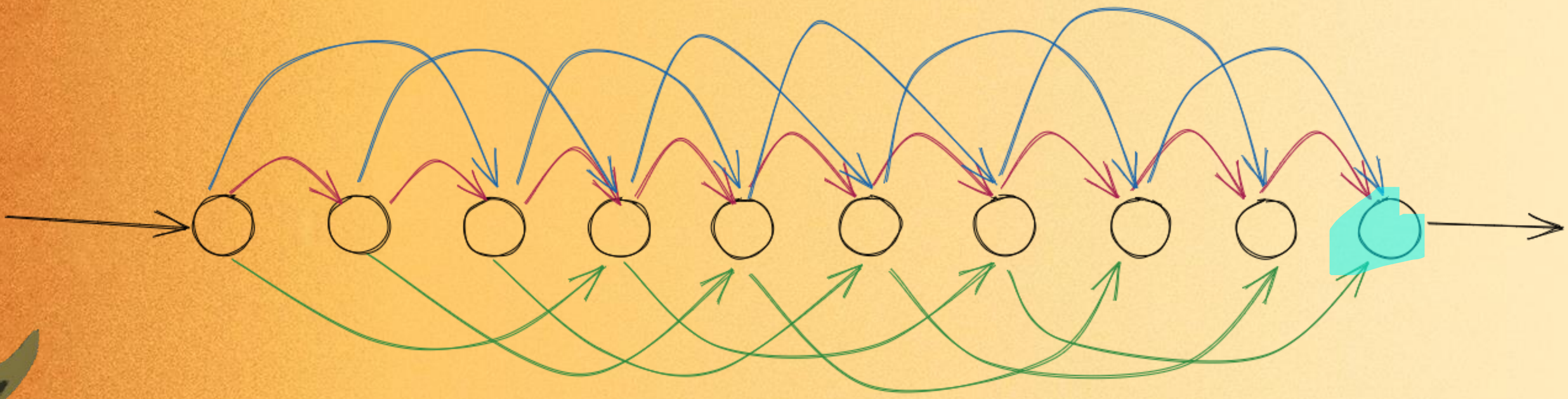
Pour 10 lieux ?

Exemple pour 10 lieux



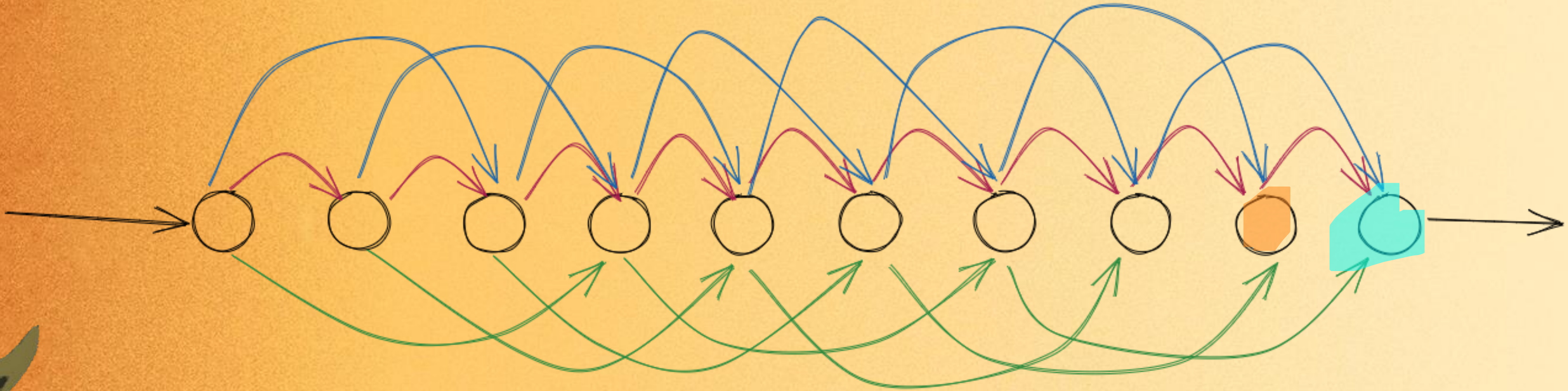
Compter le nombre de chemins possibles

Exemple pour 10 lieux



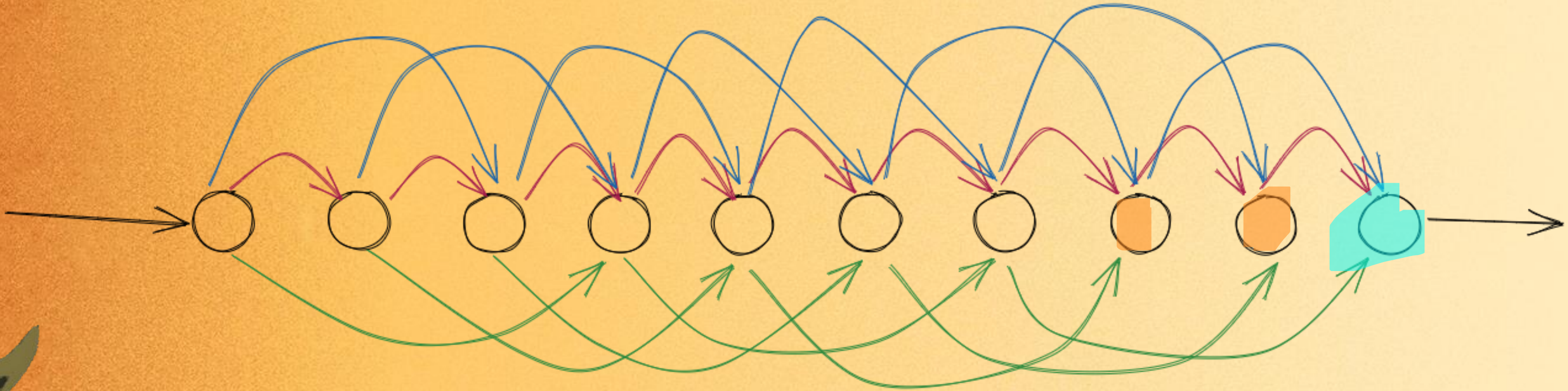
Compter le nombre de chemins possibles

Exemple pour 10 lieux



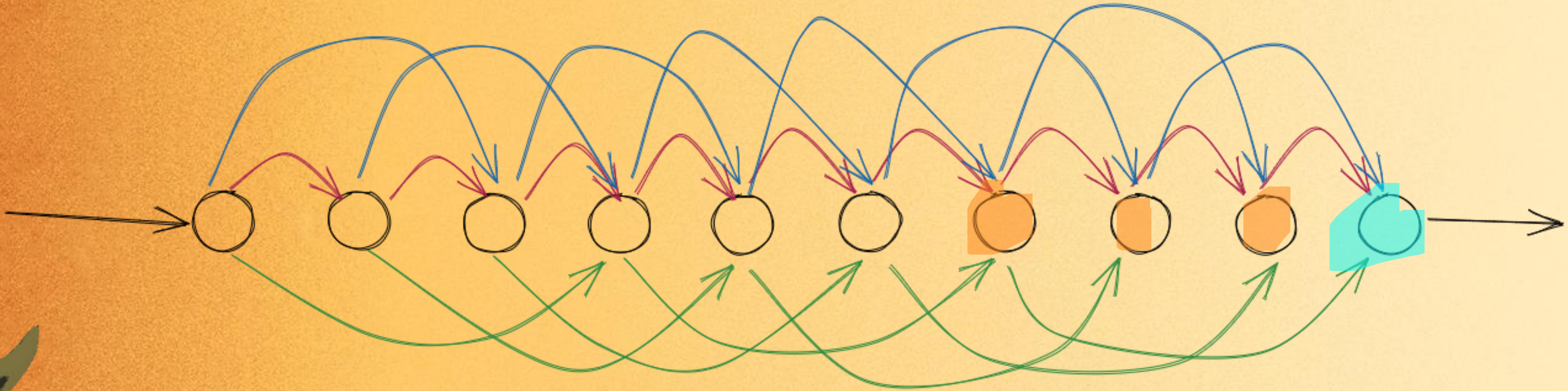
Compter le nombre de chemins possibles

Exemple pour 10 lieux



Compter le nombre de chemins possibles

Exemple pour 10 lieux



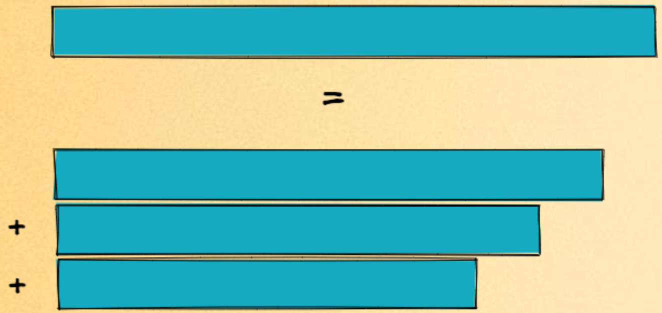
Compter le nombre de chemins possibles

DEVFEST NANTES 2022

Méthode Naïve

f = Nombre de chemins à la profondeur n

f(10) =	f(9)	En 1 lieu / minutes
	+ f(8)	En 2 lieux / minutes
	+ f(7)	En 3 lieux / minutes



Equation de Bellman

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$o(0) = 1$$
$$o(1) = 1$$
$$o(2) = 2$$
$$o(3) = 4$$
$$o(4) = 7$$
$$\ldots$$
$$o(n) = o(n-1) + o(n-2) + o(n-3)$$

Récursion & Mémoisation

Récursion avec cache

```java
Cache<Integer> cache = new Cache<>();
public int computeMemo(int n) {
    if (cache.doesntContains(n)) {
        if (n < 0) {
            cache.memo(n, 0);
        }
        else if (n == 0) {
            cache.memo(n, 1);
        }
        else {
            cache.memo(n, computeMemo(n-1) + computeMemo(n-2) + computeMemo(n-3));
        }
    }
    return cache.get(n);
}
```

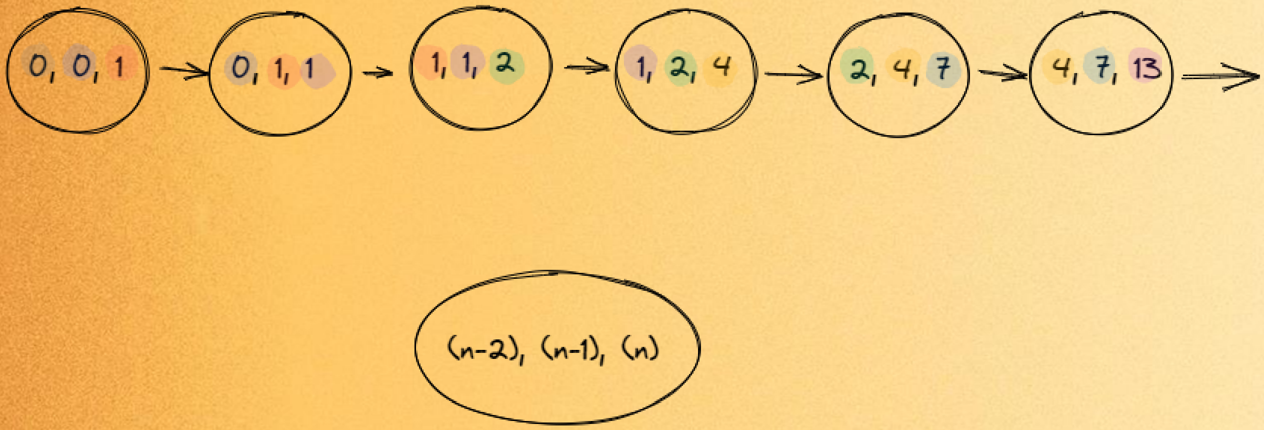
Complexité

On parcourt tous les éléments de 1 à n

Soit N

(en temps et en espace)

Parcours de Graph



Le graph des états

Analyse

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$(n-3) \leftarrow (n-2)$$
$$(n-2) \leftarrow (n-1)$$
$$(n-1) \leftarrow (n-1) + (n-2) + (n-3)$$

Tabulation

3 variables suffisent

```
public int compute(int n) {
    int n_1 = 1; int n_2 = 0; int n_3 = 0;
    for (int i = 0; i< n; i++) {
        int t = n_1 + n_2 + n_3;
        n_3 = n_2;
        n_2 = n_1;
        n_1 = t;
    }
    return n_1;
}
```

Total pour 10 Lieux

A) 5768

B) 274

C) 81

Total pour 10 Lieux

15 A) 5768

B) 274

8 C) 81

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$(n-3) \leftarrow (n-2)$$
$$(n-2) \leftarrow (n-1)$$
$$(n-1) \leftarrow (n-1) + (n-2) + (n-3)$$

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$\begin{aligned}(n-3) &\leftarrow (n-2) \\ (n-2) &\leftarrow (n-1) \\ (n-1) &\leftarrow (n-1) + (n-2) + (n-3)\end{aligned}$$

$$\begin{aligned}(n-3) &\leftarrow 0*(n-1) + 1*(n-2) + 0*(n-3) \\ (n-2) &\leftarrow 1*(n-1) + 0*(n-2) + 0*(n-3) \\ (n-1) &\leftarrow 1*(n-1) + 1*(n-2) + 1*(n-3)\end{aligned}$$

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$(n-3) \leftarrow (n-2)$$
$$(n-2) \leftarrow (n-1)$$
$$(n-1) \leftarrow (n-1) + (n-2) + (n-3)$$

$$(n-3) \leftarrow 0 * (n-1) + 1 * (n-2) + 0 * (n-3)$$
$$(n-2) \leftarrow 1 * (n-1) + 0 * (n-2) + 0 * (n-3)$$
$$(n-1) \leftarrow 1 * (n-1) + 1 * (n-2) + 1 * (n-3)$$

Un pas de plus ...

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$\begin{aligned} (n-3) &\leftarrow 0 * (n-1) + 1 * (n-2) + 0 * (n-3) \\ (n-2) &\leftarrow 1 * (n-1) + 0 * (n-2) + 0 * (n-3) \\ (n-1) &\leftarrow 1 * (n-1) + 1 * (n-2) + 1 * (n-3) \end{aligned}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} (n-3) \\ (n-2) \\ (n-1) \end{bmatrix} = \begin{bmatrix} (n-2) \\ (n-1) \\ ((n-3) + (n-2) + (n-1)) \end{bmatrix}$$

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} (n-3) \\ (n-2) \\ (n-1) \end{bmatrix} = \begin{bmatrix} (n-2) \\ (n-1) \\ ((n-3) + (n-2) + (n-1)) \end{bmatrix}$$

$$[A] * ([A] * [B]) = ([A] * [A]) * [B]$$

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} (n-3) \\ (n-2) \\ (n-1) \end{bmatrix} = \begin{bmatrix} (n-2) \\ (n-1) \\ ((n-3) + (n-2) + (n-1)) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}^n * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \blacksquare \\ \blacksquare \\ f(n) \end{bmatrix}$$

Un pas de plus …

$$f(n) = \begin{cases} 0 & si\ n < 0 \\ 1 & si\ n = 0 \\ f(n-1) + f(n-2) + f(n-3) & si\ n > 0 \end{cases}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}^n * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{[redacted]} \\ \text{[redacted]} \\ f(n) \end{bmatrix}$$

$$x^n = \begin{cases} x & si\ n = 1 \\ (x * x)^{\frac{n}{2}} & si\ n\ est\ pair \\ x * (x * x)^{\frac{n-1}{2}} & si\ n\ est\ impair \end{cases}$$

Exponentiation rapide

Exponentiation

BigInteger pour entier à précision arbitraire

```java
public BigInteger computeMatrix(BigInteger n) {
    Matrice reference = new Matrice(new int[][]
        {{ 0, 0, 1},
         { 1, 0, 1},
         { 0, 1, 1}});
    Matrice power = reference.pow(n);
    return power.at(2, 2);
}
```

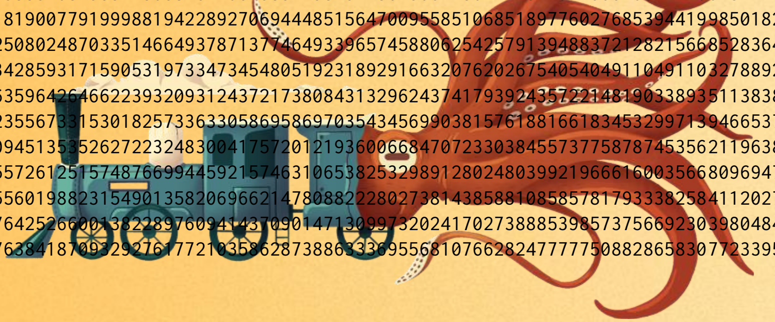
Complexité

Exponentiation rapide, on divise N par 2 à chaque étape

Soit : $log_2(N)$

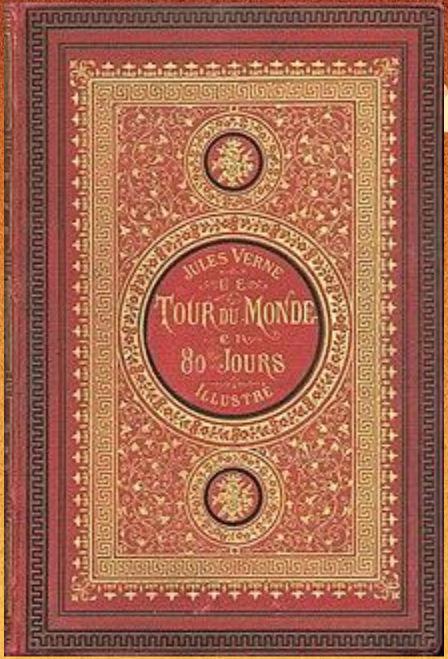
Total pour 20 000 Lieux

602772120272989212879498765925827421074931000544111889691480822268068517563053962465859871505535420787321029214246417582317464833332906986471604690451443575564622732763277782865
021647584157886757861102947250782002821499248946917035567392214576501438673071855903233281386990892777517889142825140392791359112636446350073511801398798250689373112365646482077
452977497618475246069638531137907228255234489284658833286468955158790470453896400161973237471507490124777115536053537377181428114029420661076297002280238267685575668272718701632
030047679108900202058170860807791804489267875987936213203334340159021451827371204186219548711174345119880420064899032302664993820784927236473685587206582186612514759508447849929
079427629648384933928024968703250424723969705766681011117899660167147536659206572499404934431582295428893782927079827856084116538904078045815034538541441180108778218466088987871
028239164362219761776684330093517503997528737608215705016611267703608152216388518845970634878036101972354550461256429328505221689573957880744510708993225170049612397773482002333
580188932525046104829596375570185613567999749141580899984539623823482051964403703747141004241187241131171310771679459481127771198986513635510080518608443773830754368295356014177
988418846803385291834083674243743525537930503727337927129446136176667222903559885945992268075177330951074617215734006048524250980353976619869587790758521347947262054948889135777
086461938000323471768125091516164497633518982320242318111320955746610685016645642511019312777951506900779305249054651871748558691584830851585054317398565069806012862514480519899
875300980707560526262617585182030182314990863173406117376780751139909023610065669816593633676881308344703607608775594482150974588167484305317907656858992923607995385863642449455
082992532003884428200509097204799600946489518644449319548983493995584737565711898423360643977615548114091085082715437799543266035869825912472154165509029398260637797899454667117
455832749822600874264547276774931249584470969448121958687977394083047214019843729293943414396573391719406892506293195001250147436650907452322216672432661100385086867003077839033
165934356245047205787588937314306113859008706431609310851891571820229830782722577192277653992668529851556199628988502569546720937643726538940203079161167545613516830212459004811
512551919775419050742278522680086557376798837734887793392736080153671765730813894486859474105042593291864133653644484303908029581059274466853990023553843539485092193772209830955
499049075278447560706387165147082731591805201816921117777522854986713299939130268624729399827391606027865551541967023710895990442216500309447761738435339199697161817629278488144
073079373207759552721180046426389770902940871673516483305212537066109349999022260275778431302483286677846632189090371268578871833986394227345637236903860470806421817353803368999
621227115862299153876501349051089616665052364318456430924801198107402442298721696080534065465883785919128398433503453834600476307779988094518753957865663133249110639204587416111
923529947794337608990490210550877410849391825640530813129516001518133285295372765028739744178767451148044558145951577665765602619748116293119329574060071779522290712157260120100
975000183659537719712265303874484393785365239180863840052030529904893924503420097811546817765596822181202515100739509836493817610303979497414976338592012620306028922022263093477
811159372652503543457929334417934182732736820745920788466449353168293317365826170625831798452186150624301472377922503705277460184174495798425515522579465992195282437395671916222
966599947141428342949859409325009419812151044435351242115718190077919998819422892706944485156470095585106851897760276853944199850182254551441513916752376664710604982525389797933
006279411385106843082388696668684582725873179332217082404725080248703351466493787137746493396574588062542579139488372128215668528364789617469353127866727540480199805532961077933
639228810334692777271228674881469474492252615166353409051534285931715905319733473454805192318929166320762026754054049110491103278892221949341610629456935163206395691177332518833
912711587707868614881855502352424513835099966501919213957663596426466223932093124372173808431329624374179392435722148190338935113838055529182388710368971236521027312538160041844
708804373184814177481008416637763386213333414796696776512423556733153018257336330586958697035434569903815761881661834532997139466537808663616050523309884791628030647842927343911
166892650057193816466986800365551458155400779132489639913909451353526272232483004175720121936006684707233038455737758787453562119638208512534426624858608649879911391207772113744
966773855081631506651259472281816419165959506827564385244655726125157487669944592157463106538253298912802480399219666160035668096947479548018415793087833577563386696528406790133
827417547449150051987120580850693626976589707729238065101855601988231549013582069662147808822280273814385881085857817933382584112027322716874033710544932590127981343373166720188
219214163661284701367770995579379802822764506555969934348376425266001382289760941437090147130997320241702738885398573756692303980484949539711179062771328470737884096804722790544
507144934631849521942962672902613435276819283537269388118176384187093292761772103586287388633369556810766282477777508828658307723395219610645999801128821255508716885369997812366
5943943329313



L'exposition

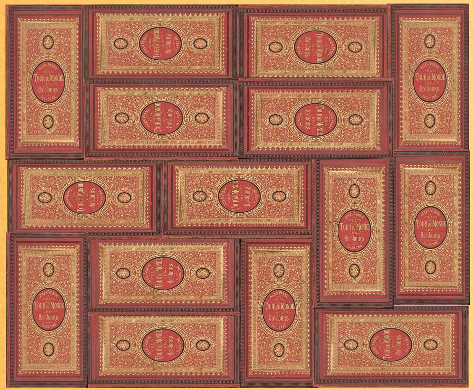
Vingt Mille Lieues sous les mers,
Voyage au centre de la Terre,
De la Terre à la Lune,
Le Tour du monde en quatre-vingts jours,
L'île mystérieuse,
...



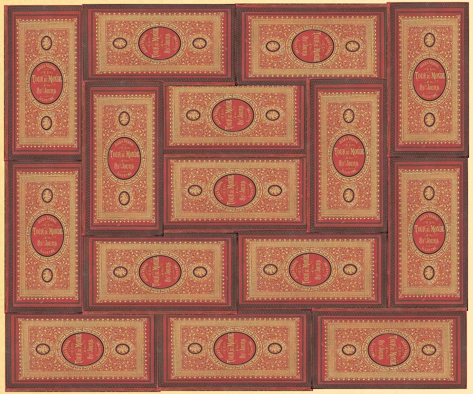
DEVFEST
NANTES 2022

Pour fêter Jules Verne, je souhaitais disposer ses livres sur de grandes tables. Les livres sont identiques de tous les côtés et font 10cm par 20cm, les tables forment un grand rectangle de 1m par 20m, de combien de façons possibles puis-je disposer mes livres en remplissant complètement l'espace ?

Exemple 5x6 :



ou



Combien ?

1) 433

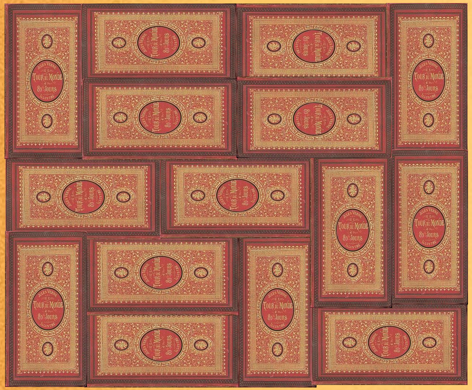
2) 1183

3) 9411

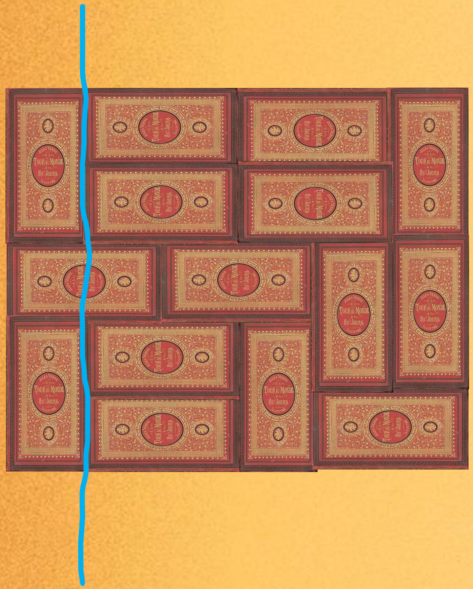
4) 75334

5x6 ?

Analyse



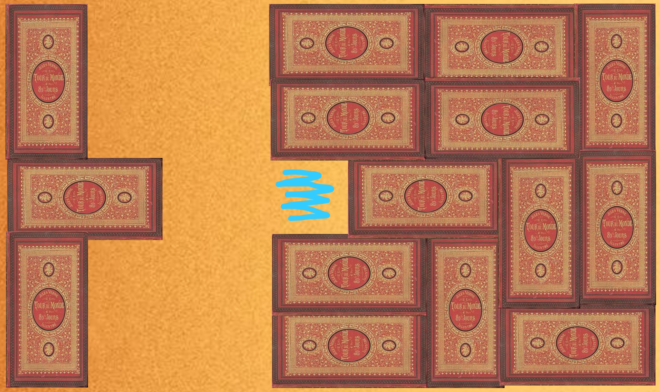
Analyse



Analyse

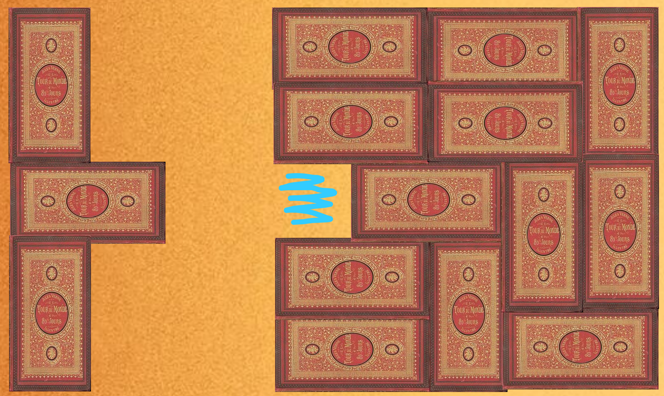
Colonne par colonne

Analyse



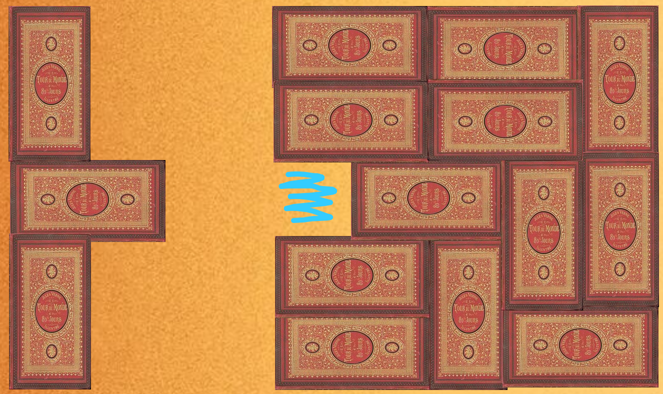
Colonne par colonne

Analyse



Problème légèrement différent

Analyse



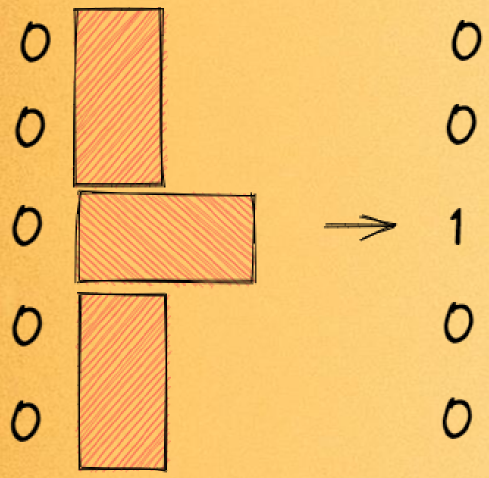
Etat plus « grand »

Problème légèrement différent

Les sous-problèmes

De la configuration *i*, calculer le nombre de possibles sur les *n* colonnes restantes

Notation

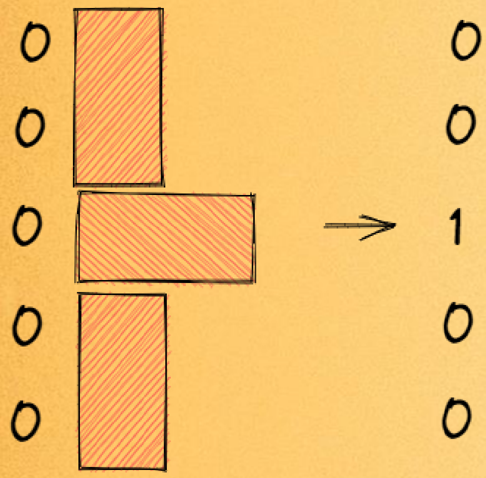


Notons 0 « case » libre et 1 « case » occupée

Une disposition transforme une suite en une autre

Notation

0

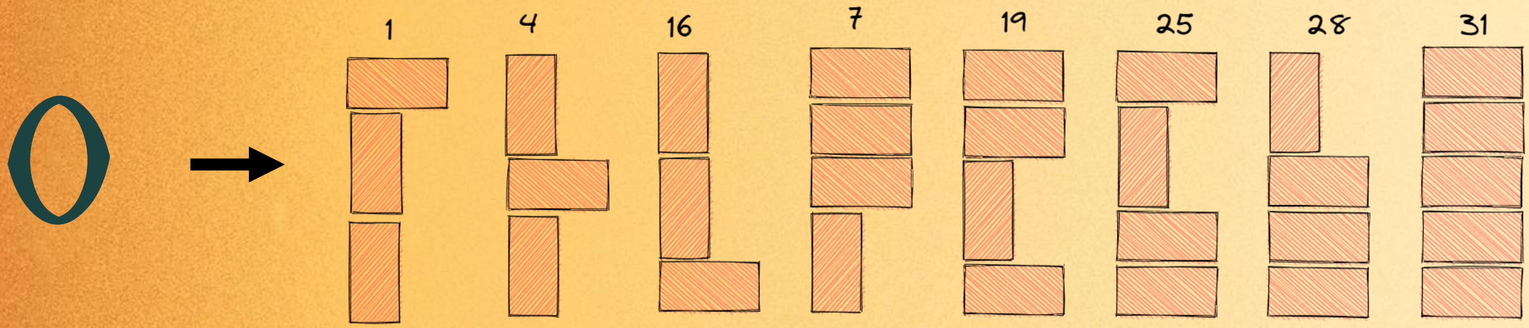


4

Notons 0 « case » libre et 1 « case » occupée => nombre en binaire

Une disposition transforme une suite en une autre

Analyse

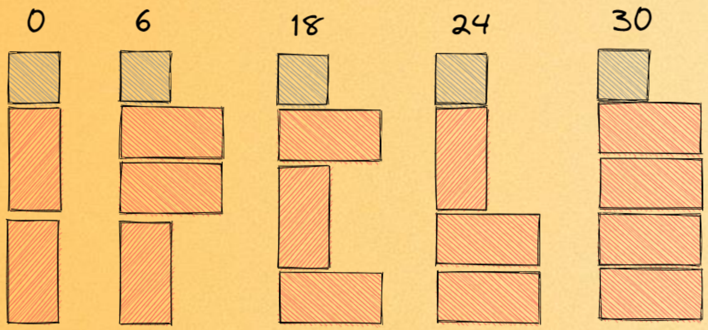


A partir de la configuration 0 :

les configurations 1, 4, 7, 16, 19, 25, 28 et 31 sont possibles

Analyse

1 →

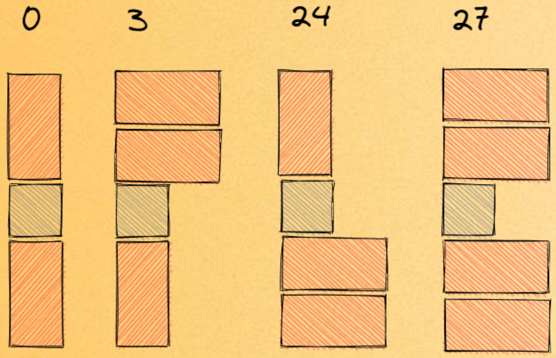


A partir de la configuration 1:

les configurations 0, 6, 18, 24 et 30 sont possibles

Analyse

4 →

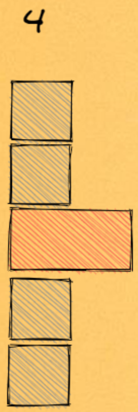


A partir de la configuration 4:

les configurations 0, 24 et 27 sont possibles

Analyse

27 →



A partir de la configuration 27:

seule la configuration 4 est possible

Deviner ? Sommer ?

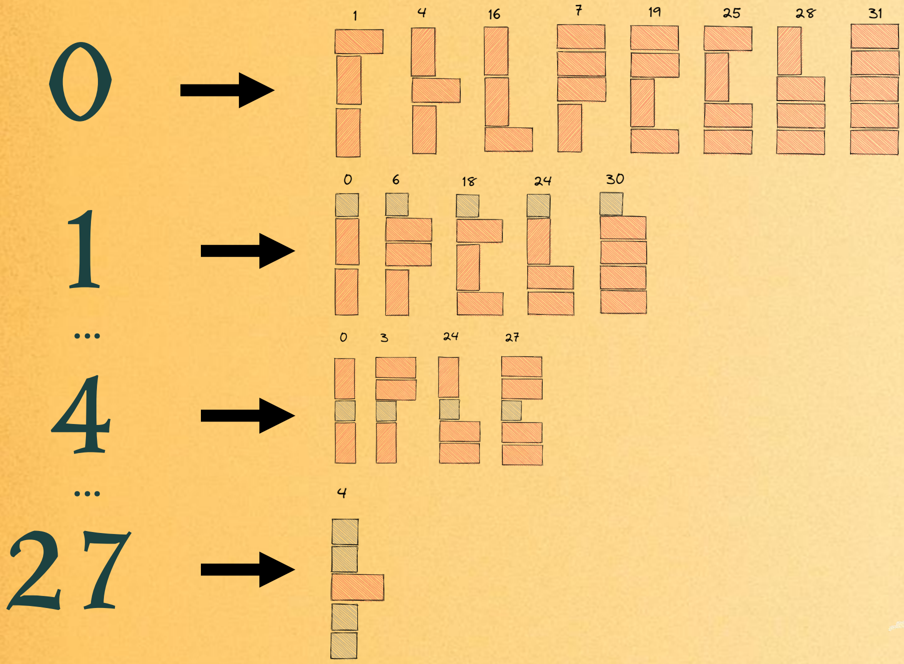
Depuis la combinaison i, et pour n colonnes restantes

L’ensemble des possibles de toutes les combinaisons accessibles pour n-1 colonnes restantes

La récurrence

$$\begin{cases} D(i,n) = \sum_{j \in succ(i)} D(j, n-1) \\ D(i \neq 0, 0) = 0 \\ D(0,0) = 1 \end{cases}$$

Analyse

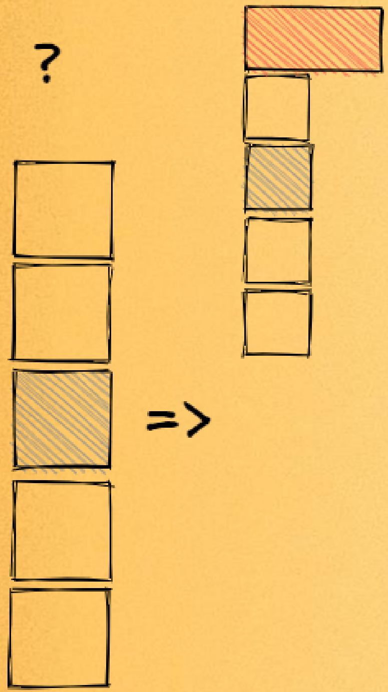


Définir les successeurs de chaque combinaison

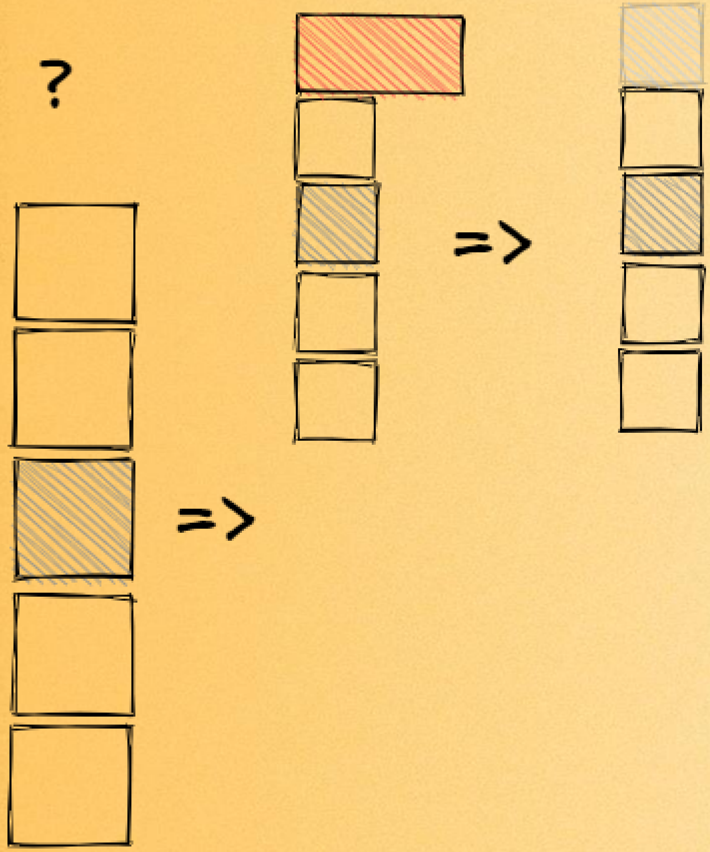
Programmation dynamique !

Encore une fois ..

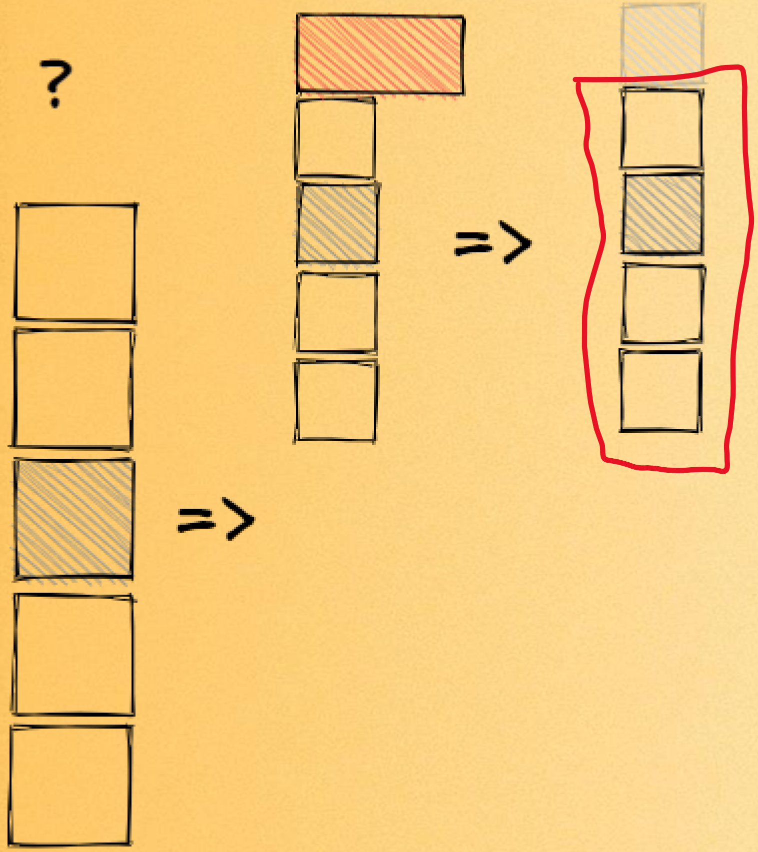
Analyse



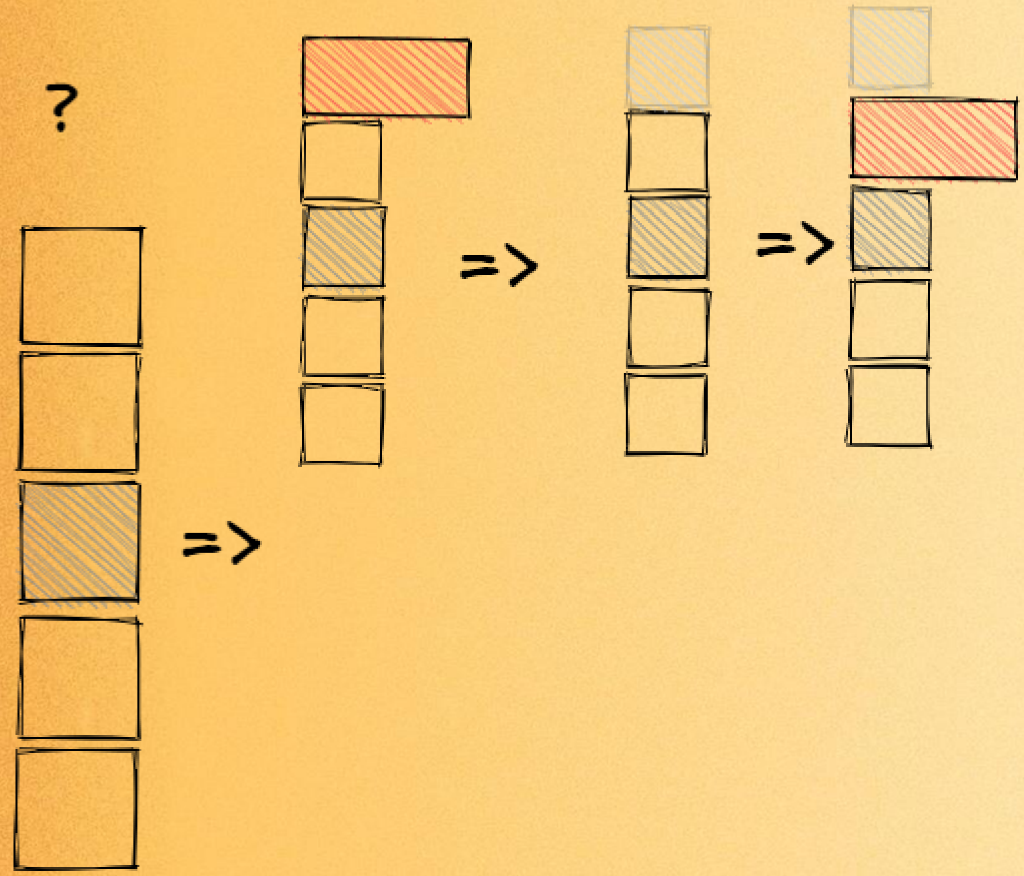
Analyse



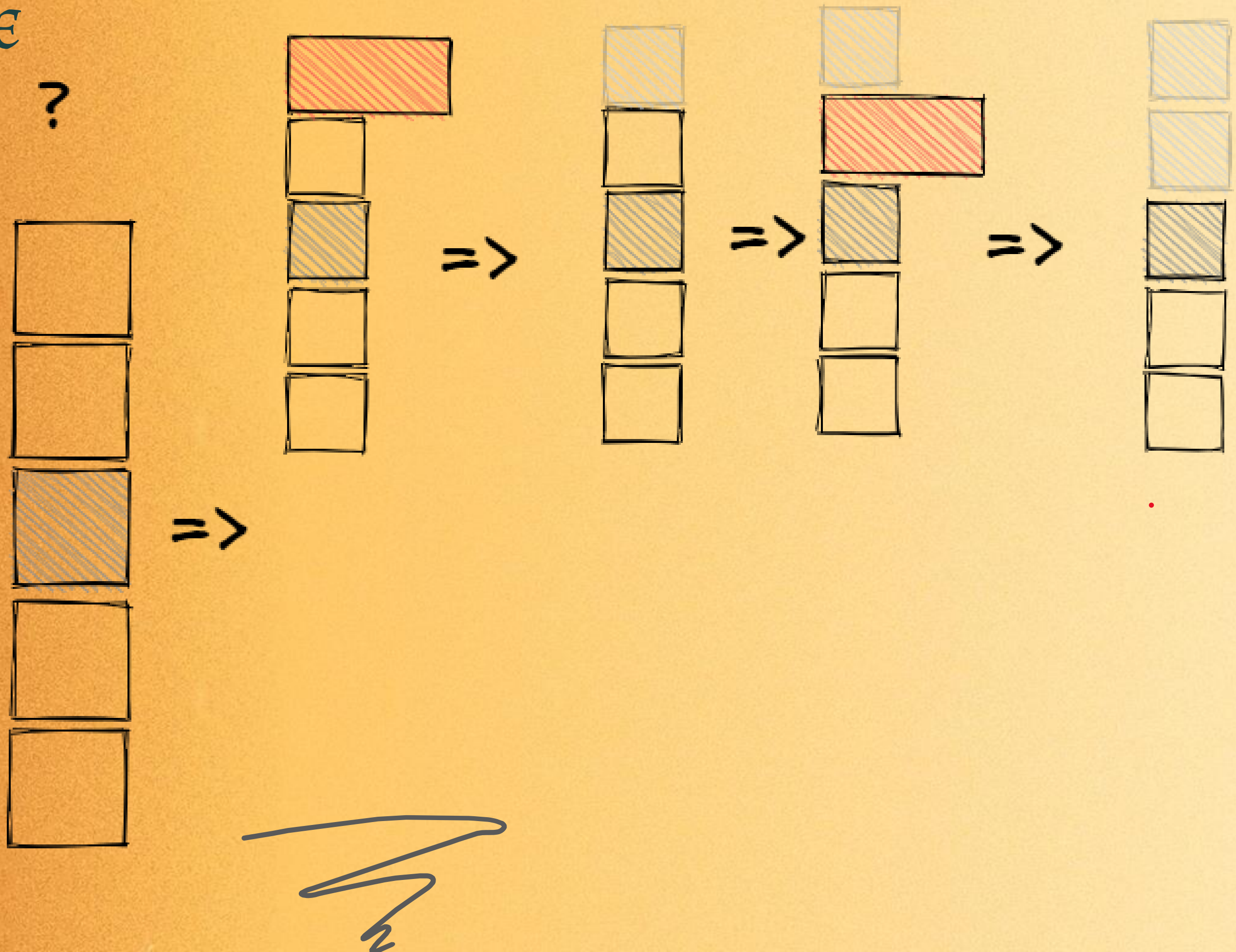
Analyse



Analyse



Analyse

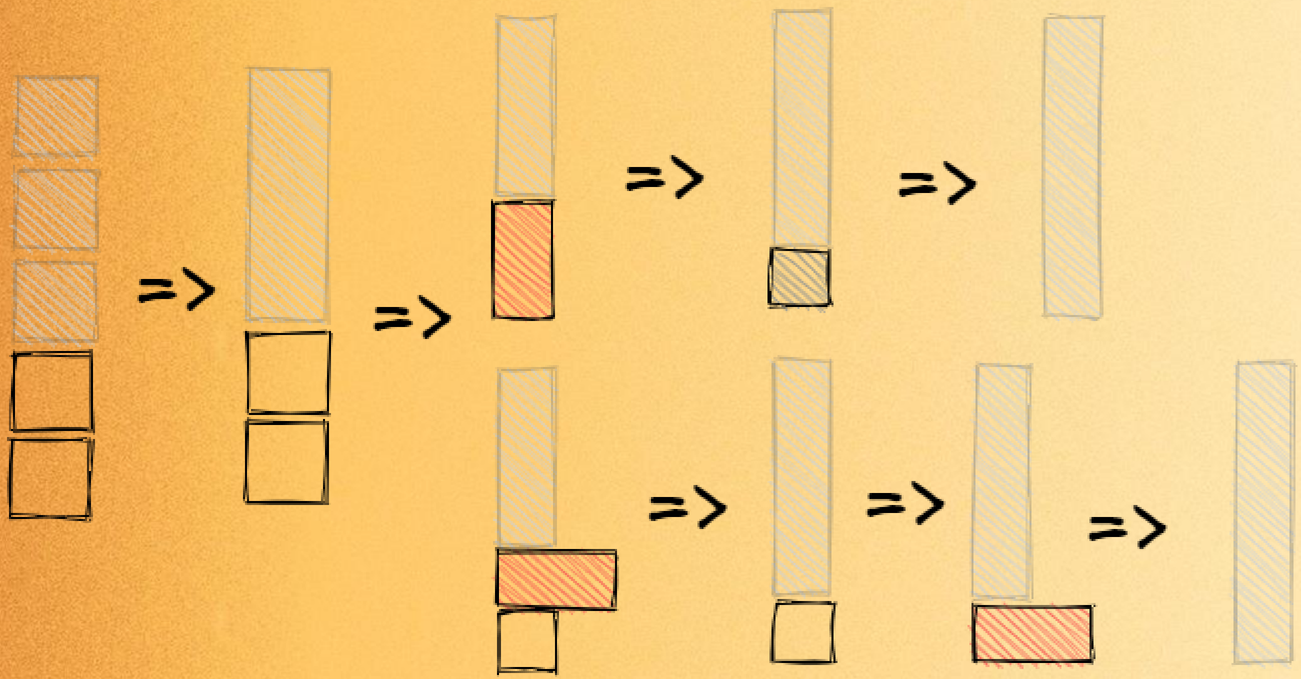


Analyse

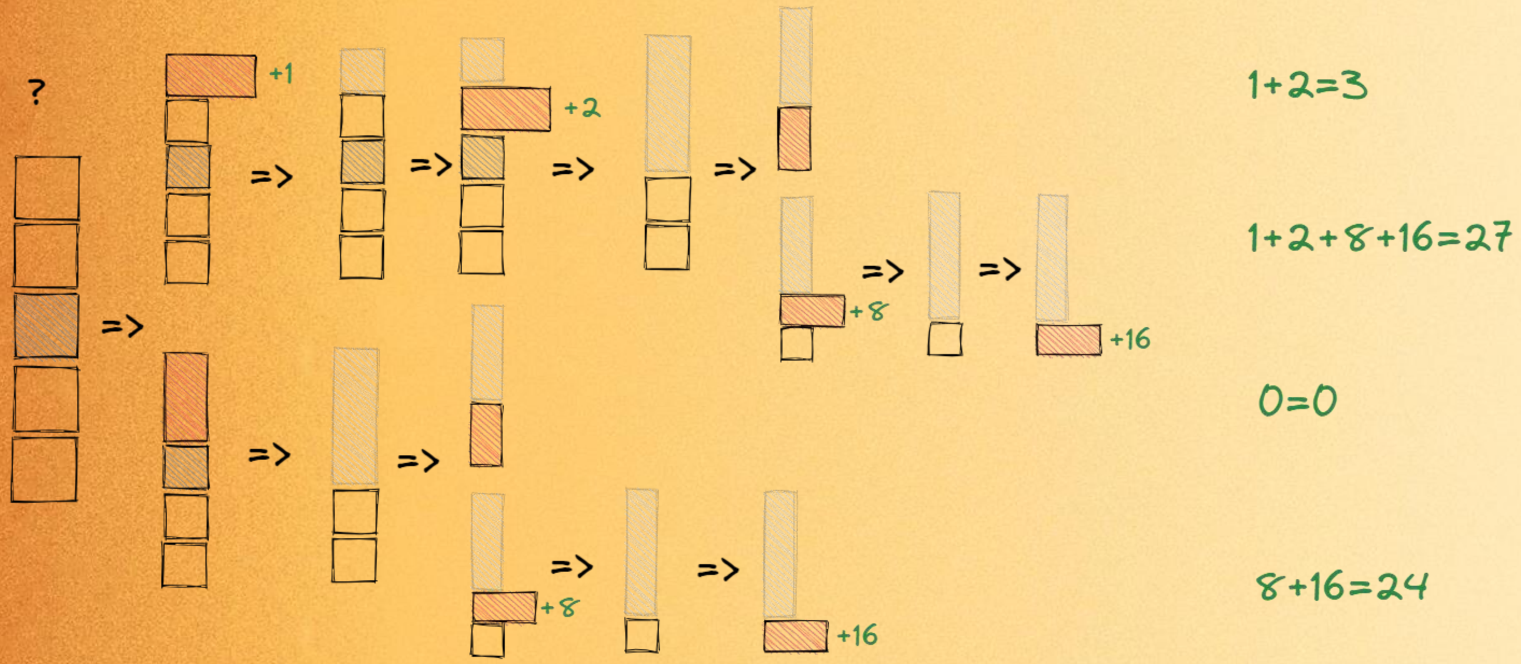
?

...

Analyse

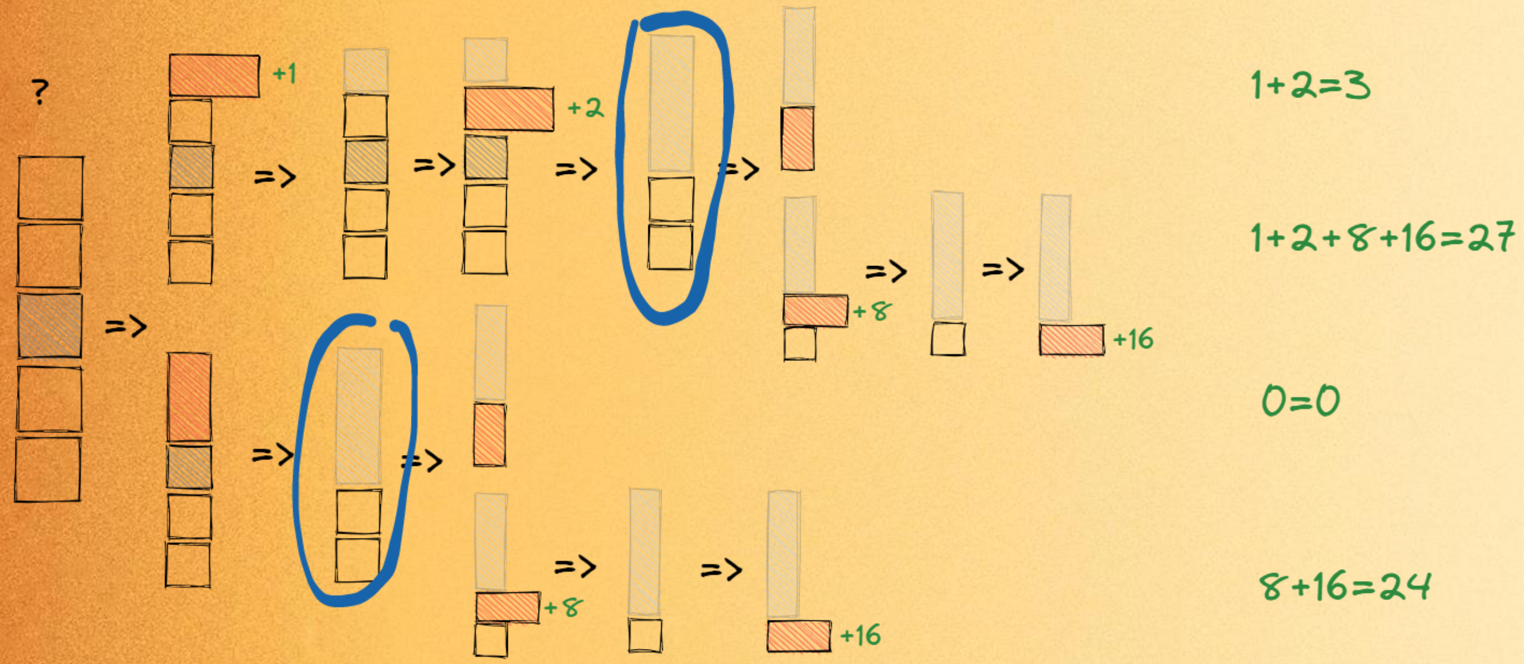


Analyse



On reconstruit les 4 successeurs

Analyse



Les sous-problèmes se chevauchent => DP!

Les sous-problèmes

Pour la combinaison i de taille n

Calculer les combinaisons de i’ de taille n’

Avec i’ la combinaison i *mise à jour* des cases consommées

Et n’ la taille restante

Deviner ? Sommer ? Combiner ?

Poser un livre à l’horizontale ou à la verticale

Union des solutions

La récurrence

$$\begin{cases} D(i,n) = \bigcup_{i'} D'(i', n-1) \\ D(i,0) = 0 \end{cases}$$

$$D'(i,n) = \begin{cases} 1 + 2 * D(i,n) & \textit{case occupée} \\ 2 * D(i,n) & \textit{case libre} \end{cases}$$

$$i' = succ(i) = \begin{cases} i/2 & \textit{horizontal} \\ 1 + i/2 & \textit{vertical} \end{cases}$$

Combinaisons

```java
public List<Integer> combinaison(int startedCombination, int size) {
    if (size == 0) {
        return List.of(0);
    } else if (startedCombination % pow(2, 1) == 1) { // case "0" occupée
        return transform(combinaison(startedCombination / 2, size - 1), 0 /* libre */);
    } else if (((startedCombination % pow(2, 2)) == 0) && size > 1) { // case "0" & "1" libre
        return union(transform(combinaison(1 + startedCombination / 2, size - 1), 0/* vertical => libre */),
                     transform(combinaison(startedCombination / 2, size - 1), 1/* horizontal => occupé */));
    } else { // case "0" libre
        return transform(combinaison(startedCombination / 2, size - 1), 1/* horizontal => occupé */);
    }
}


private List<Integer> transform(List<Integer> nexts, int nextColonne) {
    List<Integer> res = new ArrayList<>();
    for (var j : nexts) {
        res.add(2 * j + nextColonne);
    }
    return res;
}
```

DEVFEST NANTES 2022

Algo récursif

```java
public long compte(int i, int n, List<List<Integer>> successeurs) {
    if (i==0 && n == 0) { return 1; }
    else if (n == 0) { return 0; }
    else {
        long res = 0;
        for (int successeur : successeurs.get(i)) {
            res += compte(successeur, n-1, successeurs);
        }
        return res;
    }
}
```

```java
public long compute(int n, int m) {
    List<List<Integer>> successeurs = new ArrayList<>();
    for (int i = 0; i < pow(2, n); i++) {
        successeurs.add(sort(combinaison(i, n)));
    }
    return compte(0, m, successeurs);
}
```

Bootstrap

Avec mémoïzation

```java
Cache<Key> cache = new Cache<>();
public int compteMemo(int i, int n, List<List<Integer>> successeurs) {
    if (i==0 && n == 0) {
        return 1;
    }
    else if (n == 0) {
        return 0;
    }
    else {
        if (cache.contains(new Key(i, n))) { return cache.get(new Key(i, n)); }
        int res = 0;
        for (int successeur : successeurs.get(i)) {
            res += compteMemo(successeur, n-1, successeurs);
        }
        return cache.memo(new Key(i, n), res);
    }
}
```

Ordre topologique

On a facilement l'ordre sur n

On peut itérer sur les i croissants

Itératif

```java
public int compteIteratif(int n, int m, List<List<Integer>> successeurs) {
    int[][] data = new int[m+1][pow(2,n)];
    data[0][0]=1;
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j < pow(2,n); j++) {
            for (int successeur : successeurs.get(j)) {
                data[i][successeur] += data[i-1][j];
            }
        }
    }
    return data[m][0];
}
```

Itératif

```java
public int compteIteratif(int n, int m, List<List<Integer>> successeurs) {
    int[][] data = new int[m+1][pow(2,n)];
    data[0][0]=1;
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j < pow(2,n); j++) {
            for (int successeur : successeurs.get(j)) {
                data[i][successeur] += data[i-1][j];
            }
        }
    }
    return data[m][0];
}
```

Etat

On ne dépend que de i-1

L'état de notre système à chaque itération est un tableau à une dimension

Etat de taille 2^n

Itératif – Espace réduit

```java
public int compteReduceSpace(int n, int m, List<List<Integer>> successeurs)
{
    int[] data = new int[pow(2,n)];
    data[0]=1;
    for (int i = 1; i <= m; i++) {
        int[] next = new int[pow(2,n)];
        for (int j = 0; j < pow(2,n); j++) {
            for (int successeur : successeurs.get(j)) {
                next[successeur] += data[j];
            }
        }
        data = next;
    }
    return data[0];
}
```

Complexité

On a 2^n « états » possibles, au plus 2^n successeurs possibles

Et m successions à suivre

Soit : $m * (2^N)^2$

Soit : **Pseudo-polynomial**

Parce que N est *petit*

Matrice

$$
\begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 \\
1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 \\
1 & 0 & 0 & 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 & 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 & 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 \\
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
\end{bmatrix}
$$

On peut décrire les successions dans une matrice

Matrice

$$
\begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 \\
1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 \\
1 & 0 & 0 & 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 & 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 & 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 \\
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
\end{bmatrix}
$$

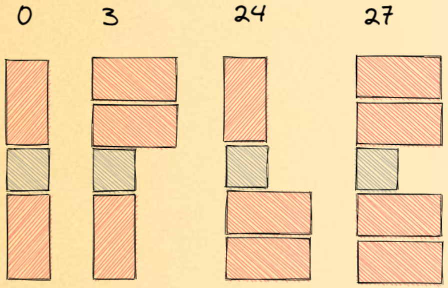
On peut décrire les successions dans une matrice

Matrice

$$
\begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 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 & 1 & 0 & 0 & 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 & 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 & 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 & 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 & 1 & 0 & 0 & 1 & 0 & 0 & 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 \\
1 & 0 & 0 & 0 & 0 & 0 & 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 & 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 & 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 \\
1 & 0 & 0 & 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 & 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 & 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 \\
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
\end{bmatrix}
$$

$4 \rightarrow$ 0, 3

24, 27



On peut décrire les successions dans une matrice

Matrice

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} * \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

Décrit les successeurs des successeurs (la multiplication)

Exponentiation Rapide

```java
public long compteMatrix(int n, int m, List<List<Integer>> successeurs) {
    int[][] matrix = new int[pow(2, n)][pow(2, n)];
    for (int i = 0; i < successeurs.size(); i++) {
        for (int j : successeurs.get(i)) {
            matrix[i][j] = 1;
        }
    }

    Matrice matrice = new Matrice(matrix);
    matrice = matrice.pow(m);
    return matrice.at(0, 0);
}
```

Complexité

On est maintenant en exponentiation rapide,

Avec une matrice de taille $2^N * 2^N$

Il y a $(2^N)^3$ multiplications

Soit : $log_2(m) * (2^N)^3$

Combien ?

1) 433
2) 1183
3) 9411
4) 75334

5x6 ?

Combien ?

1) 433
2) 1183
3) 9411
4) 75334

5x6 ?

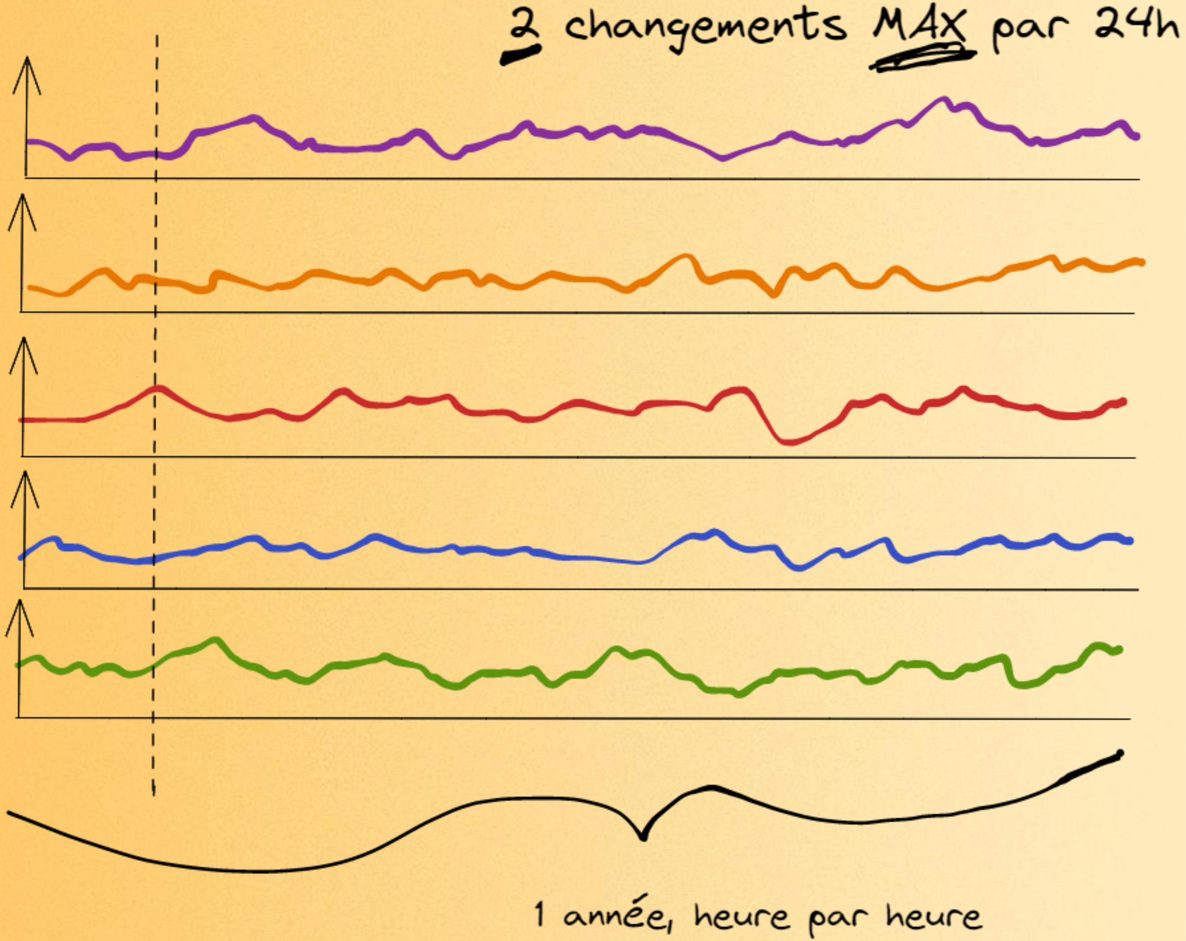
Total 1m * 20m

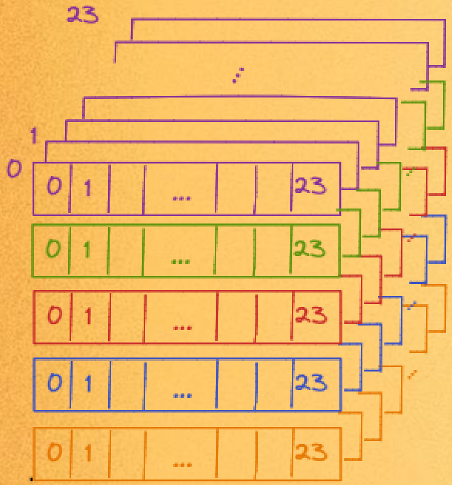
4565534236747621457009768967437352786826861072492521624579847331043213860906733870838023606447868312484137672069740511393657127500082031702465119532276842526693676152662852568352492504167432331513653482661665333834311720647583316647383055781

Épilogue

DEVFEST
Nantes 2022

Quel est le coût minimum ?





ÉTAT

Je suis sur la courbe *i*,
et mes derniers changements
étaient « il y a x:0..24 et y:0..24 »

$$
\begin{cases}
e(n,i,x,y) = e(n-1,i,x+1,y+1) + c(n,i) \\
e(n,i,x,23) = min_{j\neq i}(e(n-1,j,x+1,0)) + c(n,i) \\
e(n,i,23,x) = min_{j\neq i}(e(n-1,j,0,x+1)) + c(n,i) \\
e(n,i,0,x) = min(e(n-1,i,0,x), e(n-1,i,1,x)) + c(n,i) \\
e(n,i,x,0) = min(e(n-1,i,x,0), e(n-1,i,x,1)) + c(n,i)
\end{cases}
$$

À MÉMORISER

- Définition des Sous-problèmes

- Définition des Sous-problèmes
- Tout tester (& garder le meilleur)

- Définition des Sous-problèmes
- Tout tester (& garder le meilleur)
- Ordre topologique | graph orienté acyclique

- Définition des Sous-problèmes
- Tout tester (& garder le meilleur)
- Ordre topologique | graph orienté acyclique
- Récurrence + Mémoïzation

- Définition des Sous-problèmes
- Tout tester (& garder le meilleur)
- Ordre topologique | graph orienté acyclique
- Récurrence + Mémoïzation

Une méthode pour trouver un algorithme

Penser aux états

Modéliser et définir les états dans nos applications
Penser en « transitions » d'états

Trouver des noms cools

Un bon nommage, ça sert au quotidien

Rien à voir

~~Programmation Orientée Objet, Fonctionnelle, Impérative, …~~

« Ceux qui ne peuvent se souvenir du passé sont condamnés à le répéter. »

George Santayana

DEVFEST
NANTES 2022

FIN

Merci

Gaëtan Eleouet

meritis

$$\begin{cases} D(i,n) = \sum_{j \in succ(i)} D(j, n-1) \\ D(i \neq 0, 0) = 0 \\ D(0,0) = 1 \end{cases}$$

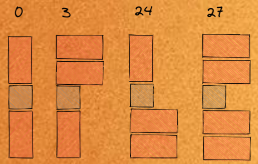
LaTeX

```java
public long compteMatrix(int n, int m, List<List<Integer>> successeurs) {
    int[][] matrix = new int[pow(2, n)][pow(2, n)];
    for (int i = 0; i < successeurs.size(); i++) {
        for (int j : successeurs.get(i)) {
            matrix[i][j] = 1;
        }
    }

    Matrice matrice = new Matrice(matrix);
    matrice = matrice.pow(m);
    return matrice.at(0, 0);
}
```

Java

https://github.com/geleouet/JulesVerne2022



EXCALIDRAW

WEBSITE
https://excalidraw.com/

WEBSITE
http://handdrawngoods.com

DEVFEST
NANTES 2022