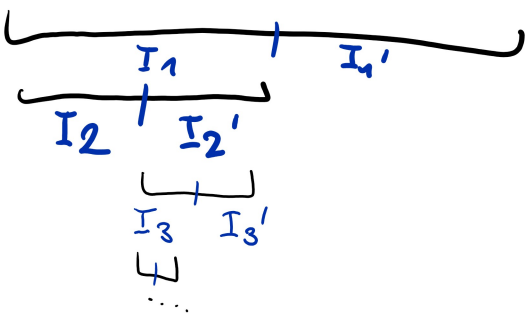
Divide & Conquer

1. Structure

- **Partition**

  divide (often by 2) espace de recherche at each step to (in the optimal case) only consider one of the subspace.

  i.e.



  Until we get a subproblem of the min size poss. to use "the trivial approach" on it.

  i.e. $\xrightarrow[m \text{ times}}$ until I_{m+1} / I_{m+1}' are trivial subpbms

- **Trivial Resolution**

  ↳ algo to solve I_{m+1}. i.e. local solution to each triv. subp.

Backtracking

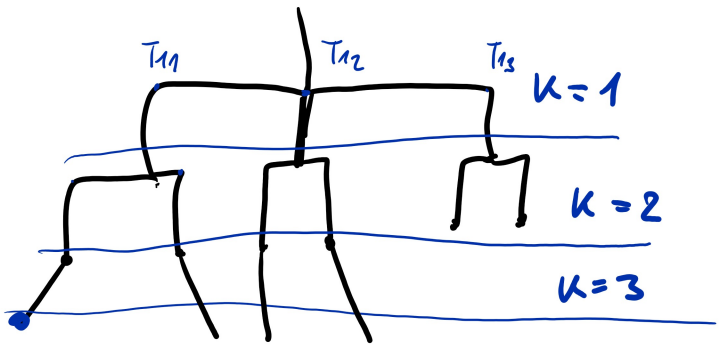
1. Structure

- Itérer (à l'étape k) sur l'ens. des Val. courantes possibles (déf par les cnds explicites T_k)

- Tester si $(x_i)_{i=0}^{k}$ est une solution locale (B_k) (ie. si c'est un candidat potentiel de sol glob.)
  - Si oui: Tester si c'est une sol glob (P)
    (⚠ il peut: $\exists$ Sol glob pour $k < N$)
    - ↳ si oui: Sol trouvée
    - ↳ Si non: passer au Step $k+1$
      (fait naturellement un arbre, k = depth)

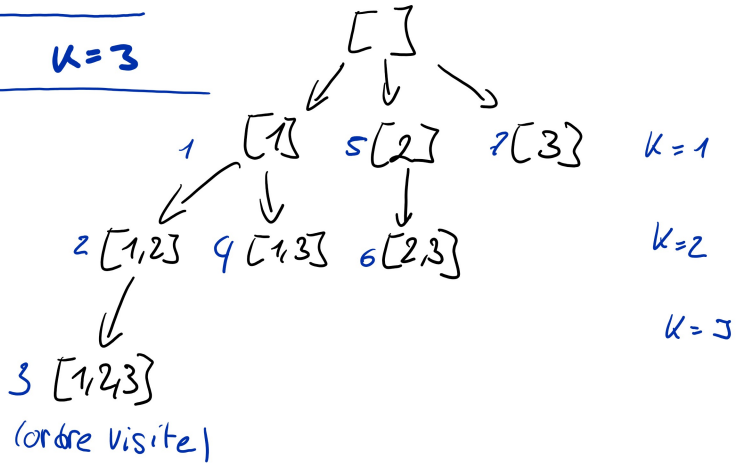
  - Si non: continuer l'itération de la Step k

- Une fois l'itération Step k finie, rec call se fini et remonte (backtrack) à celle (en cours) de la step $k-1$



N = 3 (ici)

e.g. powerset de [1,2,3]



(ordre visite)

2. Résolution cas général

```
def BT(...):
    def T(X, k, N):          # déterminé d'après X[0:k]
    def B(X, k, N):          # déterminé d'après X[0:k+1]
    def P(X, k, N):          # déterminé d'après X[0:k+1]
    out = None
    def rBT(X, k, N):
        nonlocal out
        for y in T(X, k, N):
            X[k] = y
            if B(X, k, N):
                if P(X, k, N):
                    out = X[:k+1]      # return solat {X[i] for i in range(k+1)}
                    return out         # exit point of rec call. 1st stmt after exit
                rBT(X, k+1, N)         # as soon as out isn't None : direct returned
                                       #   ↳ break for & call stack
                if out is not None: return out

    return rBT([None]*N, 0, N)
```

NB : pour return toutes les solutions :
enlever les return out / if out is not ... et
remplacer par "out.append(X[:k+1])". Ici out sera

Dynamic Programming

1_ Intuition

- Overlapping Subproblems
- Optimal substracture
- Induction

2_ "Structure"

1. We solve smallest (trivial) subproblem optimally "k=0" (basis step)
2. Then, by going through (all) the $(i \in [0, k])$ previous solutions and analyzing them, find how we can guarantee building the optimal solution for step k+1 "$k \xrightarrow{?} k+1$" (Induction Hypothesis)
3. With that relation, implement finding the optimal solution for any step k+1, by recursively resolving and storing the answer to subprob $[0 \ldots k]$ into a matrix. (Inductive step)

"$\forall k \geq 0: s(k) \rightarrow s(k+1)$"
"$\forall k \geq 0: s(k)$"

Example : "All pairs shortest path problem" (Floyd Algo)

Given a DWG (directed weighted Graph)
find shortest path between every vertices (v_i, v_j) in G.

$G = (V, E)$ $\quad V = [\![1, \underbrace{n}_{=|V|}]\!]$ $\quad L \in \mathbb{M}_{n \times n}$, $L(i,j) = \begin{cases} 0, & i=j \\ \infty, & \text{no edge} \\ \omega(i,i), & \text{otherwise} \end{cases}$

$\omega(i,j) :=$ weight of edge (i,j).

$D \in \mathbb{M}_{n \times n}$, $D(i,j)$: shortest path from v_i to v_j

things to check:

- Opti Sub prop. holds? (ie. if X is $\overset{\text{opti}}{\text{sol}}$ for k, then X holds the opti sol for all sub $0 \dots k-1$)

  ↳ Assume p is the shortest path from v_i to v_j, (ie. p is optimal $= D(i,j)$)

  if $v_k \in p \rightarrow (v_i, \dots v_k)$ is optimal $\wedge$
  $(v_k, \dots, v_j)$ is optimal $\quad (p = (v_i, \dots, v_k, \dots, v_p))$

  ($\exists$ shorter path from $v_i \rightarrow v_k$ ie.
  $D(i,k) \neq (v_i, \dots, v_k) \rightarrow \exists p', v_k \notin p' \wedge \underbrace{p' = D(i,j) < p}_{\text{contradiction}}$)

- Triv resol? ("$k=0$") Basis Step

(B.S.) Here we can just initialize D to L since the B.S. will be all the path for all adjacent nodes (ie. if $(i) \xrightarrow{x} (j)$ then the shortest path is just "given by the edge" (length 1, $\omega = x$)).

↳ "skipped"

Approach:

(I.H) _ Assuming we filled

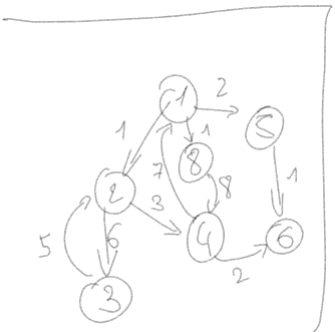
find a recurrence relation to compute (Strong induction)

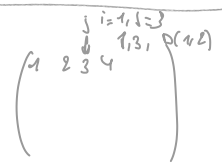
here a step k doesn't rep state of an int but a matrix D

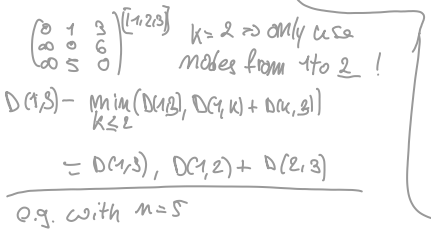
ie. $D_0, D_1 \dots$ between 2 steps we iterate through whole matrix.

And at Step k, we stored the optimal path from i to j

(for all pairs (i,j) i.e. whole mat) using only nodes from [1 to k].







ie. at Step k+1 we check for all paths from i, to j

if its shorter to pass by k+1 instead. i.e. if the path from

i to k+1, and from k+1 to j is shorter than the current

one from i to j.

We don't have the pblm "where to put the stop to k+1" bc we rec store the crt best path from $i \to k+1$ and $k+1 \to j$ and we completely replace former by this one

s.p between i,j at Step k is min length of all path

Starting from i and ending in j, using only nodes 1 to k.

Let S_{ij} the final optimal path from i to j

At each Step k (for whole mat ie. $\forall (i,j) \in N^2$):

- either $v_k \notin$ optimal path S_{ij} : change nothing in D(i,j)
- $v_k \in S_{ij}$: $D[i,j] = D[i,k] + D[k,j]$

So instead of implementing smth to check wether $v_k \in S_{ij}$ we just take $\min(D[i,j], D[i,k] + D[k,j])$ and repeat for all k. (So also for all (i,j))

The rec relation becomes:

$$\begin{cases} D_{k+1}(i,j) = \min_{\forall (i,j) \in [\![0, n-1]\!]^2} (D_k[i,j], D_k[i,k+1] + D_k[k+1,j]) \\ D_0 = L \in \mathbb{M}_n \end{cases}$$

$\Rightarrow$ Exactement comme le COINS problem:

Comment obtenir le path qu'on veut

en combinant, seulement tous les opti paths qu'on a déjà trouvé ? => Au début seulement avec les edges $(x, x+1)$

est-ce que faire $x \to k \to x+1$ plus court que $x \to x+1$!

puis on a des paths de + en + long et on peut limiter $k-1$ et restart

It's not bc D(x,y) contains a value that its the s.p. it is only if k=n ie. end of algo (recursion. D(x,y) can contain "temporary best" value that will get corrected later

Because the S.p. between i,j is fully correct only at step k=n,
all the paths between pairs are built (Checked for optimality) gradualy
ie. up to step k for each k.
Meaning the cost $D_k(i,j)$ is not optimal for $k < n$ but the path being
constructed in P (array of S.p.) ~~is optimal~~! (up to k). (no thats why we iterate through whole mat at each step)
The "real" indaction is on the paths of length k being constructed
at the same "level" for each (i, j). And since the $P_k(i,j)$ isn't complete then
its cost $D(i,j)$ is wrong,

At each step k we traverse the entire matrix to see if
passing by k on the road from i to j is shorter (for all i,j bc whole mat)

Shortest path between 1, k is D(1, k), the s.p. (short. path.)

between (k, k+1) is either (k, k+1) (if it exists and is the min)

or $\min_{\forall j}(d(k,j) + d(j, k+1))$, i.e. the min of the length of all path

starting from k and ending to k+1. (i.e. trying all the

intermediary steps possible and taking the best option after having

seen all of them).

B&B

Structure

1 : listOfChildren (getChildren) (node : Node)
→ Set

2 : cost (ĉ) ⚠ Pas de Node en arg!
(va être appelée ds constructeur)
⇒ (val, path, cost, parent) (signature)

3 : addToLiveNodes (maintain PQ structure)
ie. sorted

4 : nextENode : just pop

5 : P (defines if a Node is a solution)

2.1 fonction de coût optimale?

pour qu'elle soit optimal ⇒ 4 prémices doivent être ok.

ie pour une liste des live nodes L et x^* une sol :
on doit avoir $\hat{C}(x^*) = C(x^*)$ et $\hat{C}(x) \leq C(x)$

Comme ça on obtient que si on maintient une priority queue sur L, l'E-node choisit a toujours le plus petit coût de L i.e.

si x^* est choisi alors

$$\hat{C}(x^*) = C(x^*) \leq \hat{C}(L) \leq C(L)$$

x^* est bien de coût min. ie est la solution optimal, (par def de C, C(root) = c(optimum))

ici $c(root) = \sum_{j=0}^{n} \min_{x \in I_j}(C(x))$, on voit bien que la somme des coût min est bien le min des coût (par déf du min).

ici C est donné explicitement, on peut juste prendre $\hat{C} = C$. → C prend pas en compte la contrainte ⇒ si $\hat{C}$ le fait