Programmation Dynamique

- Une méthode générale de calcul de plusieurs valeurs dont la plupart dépendent d'autres
- Des cas où un algorithme simple naïf serait troup coûteux
- Appliquer le principe évident qu'il est inutile de calculer la même chose deux (ou plusieurs) fois
- Stocker les résultats déjà calculés dans un tableau

Un TRES petit exemple: la suite de Fibonacci

- Nous avons vu que la méthode simple récursive n'est pas bonne

- Mieux: calculer la suite jusqu'à la valeur souhaitée en utilisant un tableau

- (Mais il existe d'autres méthodes encore plus performantes)

Les coefficients binomiaux

Le nombre de façons de choisir m objets parmi n.

Ou on laisse le premier objet , ou on le prend:

$Bin(n, m) = Bin(n - 1, m) + Bin(n - 1, m - 1)$

sauf dans les cas $n = m$ ou $n = 0$

Le nombre de partitions d'un entier

Le nombre de façons de diviser un ensemble de n objets identiques en (1 ou) plusieurs parties.

Pas évident comme calcul; pas de récurrence directe.

Mais si j'ajoute un deuxième paramètre la taille de la plus grande partie je trouve une récurrence:

$Part(n, max) = \sum_{i=1}^{max} Part(n - max, i)$

sauf dans les cas $n \leq max$

La méthode naïve

aurait été de calculer directement de la récurrence (peut-être par une fonction récursive) sans utiliser le tableau;

le temps de calcul aurait été déterminé par le nombre de feuilles dans l'arbre de tous les calculs possibles

sans doute des siècles de calcul

que la programmation dynamique réduit à une fraction d'une seconde.

Un exemple (un peu) moins mathématique faire l'appoint

Je veux calculer le nombre de façons de faire un total de N centimes étant donné le nombre de types de pièces qui existent (n) et le nombre que je possède et la valeur en centimes de chaque type (tableaux Nombre et Valeur indicés de 1 à n.)

Calcul de $NBFac(Tot, i)$ le nombre de façons de faire un total de Tot centimes en n'utilisant que les pièces des i premiers types:

```
t:=0;  // sera le nombre souhaite
u:=0;  // le nombre de pieces de type i utilisees
repeter
   t:=t+NBFac[Tot-u*Valeur[i],i-1];
```

```
   u:=u+1;
jusqu'a u>Nombre[i] ou u*Valeur[i]>Tot
finrepeter;
```

sauf les cas $i = 0$ ($NBFac[Tot, 0] = 1$ ou 0 selon si $Tot = 0$)

Un exemple avec temps et probabilité: lancer des dés pour finir avec 100 points

Je lance des dés 20 fois; à chaque coup, je gagne le nombre de points sur la face visible du dé; je veux finir avec exactement 100 points.

Le calcul: la probabilité de réussir

Je calcule pour chaque p (nombre de points entre 0 et 100) et chaque c (nombre de coups entre 0 et 20) ma probabilité de gagner s'il me faut encore p points quand il reste c coups, disons $P(p,c)$.

Sauf si $c = 0$, probabilité $= \sum_{i=1}^{6} P(p-i, c-1)/6$

Avec correction pour les cas où on essaie de calculer une valeur avec $p < 0$!

Le problème du sac à dos discret

Etant donné un sac à dos et plusieurs objets (numérotés de 1 à n) de poids et valeur éventuellement différents, quelle est la valeur maximum d'un ensemble de ces objets qui ne dépasse pas ma capacité de le porter?

Supposons que je considère les objets en ordre et décide, pour chacun, si je le prends ou non;

quand je considère l'objet numéro i, la valeur de ce que je pourrai choisir parmi les objets numérotés $(i + 1,\ n)$ dépend de la partie non utilisée de ma capacité

je calcule pour chaque i et chaque capacité (entre 0 et ma capacité initiale) une valeur maximum

Le tableau me donne immédiatement la réponse à la question
et facilement un ensemble qui réussit cette valeur maximum.

Un algorithme très efficace si les poids sont des entiers et ma capacité n'est pas trop importante (exprimée dans la même unité)

Le problème du voyageur de commerce

Il veut visiter toutes les n villes où il a des clients et rentrer chez lui
sans passer deux fois par la même ville
et dans le temps (ou coût ...) le plus petit possible.

Le temps pour visiter les derniers i villes dépend de quelles villes elles sont et d'où il se trouve après la $(n-i)$ième visite

forme générale de la récurrence:
$V(E,i) \;=\; min_{j \in E-i}(c(i,j) + V(E-i,j))$

La programmation dynamique ne donne pas d'algorithme très performant mais aucun meilleur n'est connu qui garantit de donner toujours le parcours optimal.

Variante numéro 1
Economiser l'espace

Quand la récurrence donne la valeur de la colonne i du tableau comme fonction de la colonne $i+1$ (ou $i-1$) on n'est pas obligé de garder tout le tableau; il suffit de garder deux colonnes à chaque moment.
(vrai pour les probabilités dans le graphe, le sac-à-dos, le voyageur)

Si, en outre, chaque élément d'une colonne ne dépend que des éléments de l'autre colonne qui sont en dessus (ou en dessous), il suffit de garder une colonne qui, à chaque moment, peut contenir un partie de l'ancienne colonne et une partie de la nouvelle!
(vrai pour le sac-à-dos et peut-être le voyageur)

Variante numéro 2
Eviter les calculs inutiles

Il se peut que quelques éléments du tableau ne sont utilisés dans le calcul final qui nous intéresse (c'est-à-dire ni directement ni indirectement) mais que la structure du problème est trop irrégulière pour savoir à l'avance lesquels; (par exemple dans le cas de la chaîne de Markov, il est inutile de calculer la probabilité de réussir en 90 transitions à partir de l'état j s'il n'est pas possible d'arriver à cet état en 10 transitions de l'état initial.)

En ce cas, la programmation dynamique comme décrite fait des calculs inutiles qui n'auraient pas été faits par la méthode récursive citée avant. Mais il est toujours vrai que cette méthode récursive fait BEAUCOUP plus de calculs parce qu'elle fait très souvent le même calcul plusieurs fois.

Il existe une méthode mixte qui ne fait que les calculs utiles et ne les fait qu'une fois: elle consiste à utiliser la structure récursive et un tableau; la première fois qu'un résultat est calculé il est inséré dans le tableau; les fois suivants, cette valeur est récupérée du tableau.

Nécessite une méthode de savoir quels éléments du tableau contiennent des résultats valables; ou une valeur impossible pour les non valables ou un tableau auxiliaire booléen.

exemple: encore une fois les listes multiplicatives mais simplement calculer le nombre (avec terminaison n) au lieu de les afficher toutes.