Programmation Dynamique

- Une méthode générale de calcul
- 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 petit exemple: un jeu contre la nature

(Equivaut à maximiser la probabilité d'être dans un état cible d'une chaîne de Markov à un moment donné de l'avenir)

Supposons que dans un graphe orienté G donné, à un instant pair je choisis l'état successeur mais à un instant impair le choix est aléatoire (et disons uniforme);
Je veux entrer dans l'état C à l'instant 100.

Mes choix à l'instant 100 sont évidents (et dépendent de l'état après 99),
donc, je peux calculer ma probabilité de réussite en fonction de l'état avant 99),
donc, mes choix à 98 sont évidents,
et ainsi de suite jusqu'à mes choix à 0.

J'utilise un tableau qui, pour chaque instant (entre 0 et 100) et chaque état me donnera la probabilité de réussite si je suis en cet état à cet instant;

Les éléments de ce tableau sont reliés par une recurrence.

Je remplis le tableau itérativement.

Le résultat souhaité est déterminé facilement du tableau final.

La méthode naïve

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

le temps de calcul aurait été déterminé par le nombre de chemins de longueur entre 0 et 100 dans le graphe;

sans doute des siècles de calcul

que la programmation dynamique a réduits à une fraction d'une seconde.

Le problème du sac à dos

Etant donné un sac à dos et plusieurs n objets de poids et valeur différents, quel est la valeur maximum d'un ensemble d'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 $(i + 1,\ n)$ dépend de la partie non utilisée de ma capacité

je calcule pour chaque i et chaque poids (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 important (exprimé dans la même unité)

Le nombre de partitions d'un entier

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

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 recurrence 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.

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!

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;

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 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.