Nom : DEGRYSE Sujet : 12 Jury : / Page : 1/3

12. Exemples d'algorithmes gloutons et de retour sur trace

Niveau : MP2I / MPI Prérequis :

Motivation : Introduction de nouveaux outils pour la conception d'algorithmes exacts ou d'approximation.

I) Algorithmes de retour sur trace

① Contexte

- Un algorithme de retour sur trace répond dans la plupart des cas à un problème de décision ou à un problème d'énumération

Exemple 1 : Etant donné un graphe G et deux sommets s et t de ce graphe :
- Décision : Déterminer s'il existe un chemin de s à t.
- Enumération : Déterminer les chemins simples de s à t.

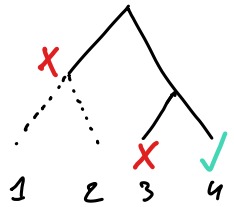
② Principe

- Pour répondre à un tel problème, un algorithme de retour sur trace construit incrémentalement une solution candidate et revient en arrière lorsque celle-ci ne peut plus être complétée en solution valide du problème

Remarque 1 : Ne pas confondre avec la recherche par force brute, qui se contente d'énumérer des solutions puis vérifie si elles sont valides.

Solutions	1	2	3	4
Validité	✗	✗	✗	✓

• Force brute



• retour sur trace

Exemple : recherche de solution de sudoku

1	2		4
		1	2
3			
	1		3

Mise en oeuvre : construction itérative d'une solution, retour en arrière.

Amélioration : ordonnancement des choix pour accélérer l'éxécution du programme.

Avantages : • facile à programmer
• Corection facilement démontrable

Inconvénient : • Grande complexité.

Cas pratiques d'utilisation :
- Solveurs de puzzles, casse-tête ...
- Analyseurs syntaxiques.
- Langage de programmation logique : Prolog.

Activité : Sudoku 9x9 / Problème des 8 dames en séance de TP.

Ne rien écrire hors du cadre.

③ Exemple d'application : SAT-Solveur

Problème (SAT-CNF) :

Entrée : une formule ℓ sous forme normale conjonctive

Sortie : ℓ est-elle satisfiable ?

Implémentation et correction de l'algorithme de Quine DEV 1

Activité 2 : Séance de TP, implémentation en Caml de l'algorithme DPLL pour le problème SAT-CNF.

La complexité du retour sur trace vient des échecs, que se passerait-il si l'on savait quel choix était le meilleur à chaque itération ?

II) Algorithmes gloutons

① Contexte

Un algorithme glouton répond le plus souvent à un problème d'optimisation, qui consiste en la recherche d'une solution maximisant (resp. minimisant) une récompense (resp. une pénalité)

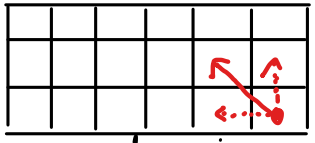
Exemple de problème d'optimisation :

. Etant donné une liste d'entiers, déterminer une sous-liste contigue de somme maximale.

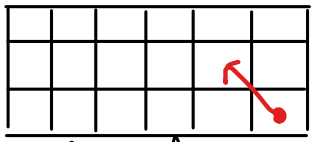
② Principe

Pour répondre à un tel problème, un algorithme glouton construit incrémentalement une solution candidate, en faisant à chaque étape un choix localement optimal, en espérant que la solution finalement obtenue soit globalement optimale.

Remarque : comme pour la programmation dynamique, un algorithme glouton peut être vu comme une division en sous-problème, selon une structure optimale, mais qui ne se ramène qu'à un seul sous-problème.



prog-dynamique



algo. glouton.

Avantages : . très rapide . Idée sous-jacente intuitive

Inconvénients : . Ne fonctionne pas pour tous les problèmes

③ Problème du choix d'activité

Définition 1 : une activité est un couple d'entiers positifs (s, f), s est l'heure de début de l'activité et f l'heure de fin.

Définition 2 : deux activités (s_1, f_1) et (s_2, f_2) sont compatibles si $[s_1, f_1[$ et $[s_2, f_2[$ sont disjoints

Problème du choix d'activité : Etant donné un ensemble $S = \{a_1, a_2, \dots, a_n\}$ d'activités, déterminer un sous-ensemble de S d'activités deux à deux compatibles de taille maximale.

Propriété 1 : Soit a_i une activité de S ayant une date d'échéance minimale. Il existe une solution du problème du choix d'activité utilisant a_i.

```
choix_activites(s, f):
    n := longueur(s)
    A := a_1
    i := 1
    pour m allant de 2 à n
        Si s[m] >= f[i]:
            A ← A ∪ {a_m}
            i ← m
    renvoyer A
```

Théorème 1 : Si les activités sont triées par date d'échéance croissante, alors choix_activites renvoie une solution du problème du choix d'activités.

④ Codage de Huffman

Définition 3 : Un codage de Huffman sur un alphabet Σ avec une fonction d'occurences $f_\Sigma : \Sigma \to \mathbb{N}^*$ est un arbre binaire dont les feuilles sont les éléments de Σ.

Définition 4 : Si C est un codage de Huffman sur Σ, son coût est défini par $\sum_{a \in \Sigma} f_\Sigma(a) \cdot h_C(a)$ où $h_C(a)$ est la hauteur de a dans l'arbre binaire C.

Problème du codage de Huffman

Entrée : un alphabet Σ et une fonction d'occurences $f_\Sigma : \Sigma \to \mathbb{N}^*$

Sortie : Un codage de Huffman C de coût minimal.

Propriété 3 : Si C n'est pas complet, alors il ne peut pas être un codage optimal

Propriété 4 : Si x et y sont deux lettres de Σ ayant le moins d'occurences, alors il existe un codage de Huffman C optimal ayant x et y comme feuilles voisines de profondeur maximale.

Propriété 5 : On pose $\Sigma' = \Sigma - \{x, y\} + \{z\}$, avec $f_{\Sigma'}(z) = f_\Sigma(x) + f_\Sigma(y)$. Si C' est un codage de Huffman optimal pour Σ', alors le codage C obtenu en remplaçant la feuille z par un noeud ayant x et y comme fils est optimal pour Σ.

⑤ Problème de couverture d'ensemble

Entrée : X un ensemble fini, $F \subseteq \mathcal{P}(X)$.

Sortie : Une couverture $C \subseteq F$ de X de cardinal minimal.

Théorème 2 : l'algorithme glouton renvoie une couverture $\ln|X| + 1$ - optimale

DEV 2