PKOJET KO- COMPLEXITE

Complencité des a**lgorithmes**

- Problème d'optim*i*sation, représenté par un programme d'optimisation

composé de fa lancian abjectif et d'un ensemble de contraintes. - Algorithme exact : permet de traiter tous les cas possibles afin

de chocsin la solution fa moins coûteuse.

→ consommation de stessources temporelles et matérielles

(suite a fiese plosion combinatóire). - Alg*orithme a*pproché : obtenir une solution non précise dans un

těmps d'*e*xécution raisonnable. Pourqu*oi la.complexité* ?

cómpanen les algorithmes solutions afin de les classifien. *C*omment *choi*si le m*eill*e*ur algorithme*?

au nivea*u de*.s*e*

couns, on s'interesse - Convergence: tower eine solution.

Qunc messounces - Validité : solution isalidee.

*t*emporelles. - Optimisation des ressources:

• spatiafes : mémoire etiliseể.

tempanelle: temps d'escécution nécessaire. *C*om*m*ent estimer *l*a *complencité* ?

- Tethode Bre perimentale (approche statistique) : exécuter te prognamm en estalisant plusieuns valeurs de données pour dosenver l'évaluation

du temps en fonction de la quantité de données (counbe). Approche Pomelle (approche théorique): le coût représente le

temps d'encécution theonique. Calcule en fonction ve ne dépend pas de l'environnement du nombre des opérations d'escécution.

e ffectués par le processeur. - De n'est pas possible de calculer la compleneité escache *: nou*s

ealeafenons sen andre de gaandeur grâce a' des notations asymptomatiques connues sous le nom de n*ot*ation *de Landac*e.

*Méthode formelle: Not*ation de *Land*a*u* :

T(n) : coût (temps d'exécution théanique). .0: majoration du pine ,cas (le plus grand nombre d'opérations *>f*ag) => 3 no. 3czo' tq Hndo f(n) se.g(n)

*f*e*s*t dominee a*symptota qu*e ment par og (magorée par *g).*

*of* est nég*ligeable de*vant g. = 2(g) + 3 no. 3.20 dq W nano ffrai) > fog(n)

.2:

g est majonée par f .: f= @(g) - 0(g) ef g = OCRA

=> 2.deR\*.30 do Home

Sig(n) f(n) is a good \* O(n)bonne supérieure > Majination du pire cas. \* 2 (n) bonne inférieure Minenation du meilleur des cas. \* @(n) banne exacte no pues précise que les précédentes. Quelques réglées à appliquer pour déterminer *fa*.*comp*le*rcité o d*'un

*a*lg*orith*me a pa*r*tir du nombre d'exécution des opérations *T(n*): - Negligen les contraintes additionnelles

*Ex*em*ple: T()= +*22 = O(n?) - Negligen fes constantes multiplicatives.

*E*xem*ple*: T(n) = CARP = O(n) Negliger les termes d'andre inférieur.

*Ex*em*pl*e: T(n*)* = SA K n? +62 m2 O(n2) + O(n) = 0(%) *- En* cas de multeplication:

Excemple: 0(ne) & (03) = o(os)

*Ch*aque tra*itement él*éme*ntai*re a *u*n coû*t* = 1 :

*-* La déclaration - L'apectator -L'apération élémentaire y ont un coût = 1

Le test - Le return Pour une boucle:

- calculer le coût total *in*terne. ..de multi*p*lier par *le* nombre d'itération.

o Co*ût total =* s*omm*e *des couts \* Lor*s *du calc*ul d*u pi*se .cas: - s'il y a a blocs (exe: waif et epse), on choisit le cout le

plus élevé. *Le*s classes de complexité : *- dl*a*g(n)) : L*es algorithmes sub-Pineaires. *. : Le*s algorithmes *l*inéaires.

-> rapide *. O(n* l*og (nl) :* Les algorithmes quasi - lineavies.) *-OC*R \*) : Les algo*rith*mes polynomiaure (634). fe*nte Lola")*: Les algorithmes ereponentiels. - impra*ticabobe.*

*C*o*ut d'un*e opération *de pui*ssance n\*:

& fois ead (1+1) fois *L*a campfencité devient TIME SA (n 949) = O(no)

c'est le cas d'un algorithme Trivial.

*Récunsivité et* comple*scit*é des *algonit*hmes récun sips

*-L*a sécunsivité permet d'écure des algorithmes plus précis et plus clairs.

o its sont plus couteurs en terme de complexité par rapport à un

algenitame itenatif. IP faut être sûr qu'on tombera toujours sur un cas d'arêt.

A On parle de récursivité terminale et non terminale que pour la récursivité I simple.

*- Recursivité terminal*e:

l'appel récursif constitue la dennicre instruction. - Récunsivitê non termin*ale :*

Il existe d'autre(s) instruction (s) aprés l'appel nécunsip *Ex*em*ple (*Récu*rsivité nos te*r*minale):*

void Algo (int n)

{ if (nso)

(Amèt lensque neo) algo (n-1);

printf ("%d", o); }} on suppose que n = 5:

pile d'exécution en ne pourra pas exécuten l'instruction printf can l'exécution de p'appel decussif n'est pas encore teaminée.

► fe.compilateur sawegarde cette valeur dans une pile d'exécution.

A la remontée, l'algorithme ua affichen 1 2 3 4 5 . Calcu*l de* la.com*plenité* des algor*ith*mes sécunsifs (Recursivité simple)

*Excemple:* Fonction puissance

int Puissance *(*min)

{ if (n=co) then

return a;

else

retuin (net Puissance (ne, n.))}

opération *de base* T(n) = 2 + T(n-1)

-5 *Calcul de l*a com*plescité d*es algor*ithm*es rêcur sifs (Récursivité multi*pl*e)

*Exe*m*pl*e: Probleme *de Hol*ubk S*uite de* Fibonacci

int Fibonacci (n) { if (nero Ii nas) = 3

return 4 : 31 else

return ( Fibonacci (n - 4) + Fibonacci (n-2)); 3. → T(n)= T(n-1) + T(n-2) + 4

appefs récursifs a' 2 parametres différents - Récursivité mullapte.

*Exemp*le : les *tours de H*an*oï (* Récu*rsivité simple)*

*som*e

*Algorithm*e

Hanoi (n, dep, int, dest)

if (n == 1) then

déplacer le disque de dep.vens dest.

else

.

- Hanoi(- 4 , dee dest., Joh)

déplacer le disque de dep. vens dest.

Hanci(n-n, int, dep, dest.) endio EndHanei

Complenité :

T(n)= Ş 1 si nen

(T(n-1) + A + Tln - A) sinon

sj F = 4

LL

Désécun*si*vation et complereité des algorithmes récunsifs

Dérécursivation: thans formen un algorithme récunsif en un afgonithorne

équivalent ne contenant pas d'appels recunsifs. *Exemple*: a est un d*iviseur de b*? *(D*énécunsivation : Réc*u*nsivité terminate

Diviseur (*i*nt n, int b)

Diviseur (int a, antb) if (a <=0) then Erreur if (a <=0) then Erreur etse

while la<b) do if (a)= b) return (a==b) Touri*st* babua. efse

end-white return (Diviseur (a, b-a)) return (a == b); EndDiviseur

End Diviseur *Déné*c*ursivali*on (*R*écunsivité non termina*le*):

*Encemple*

Factoriel (int n) Factoriel (int n)

Pile init() if (nero)

while (n=1) do return Ai

Pile. push (on)

nunc . return \* Factoriel (n-a); end while End-Fachaniel

Fel; while (not Pile. empty) do

Pile.pop(n)

F-F#ni End-while Return F; End-Algorithme

else

**Denēc*unsive*r**

Résolution de l'équation de récurrence sans second membre :

| Tan) = X Ta - 1) + Xi TKP - 2) + ... ( k o – 8 )

6 Tin) an T(n-1) + X2 T(0-2)... ax T(n-k) = 0 1. Associer un polyname caractéristique d'andre f

1 P(se) = nk kark-a - xan k-2 -... kr. 2. Chercher fes racines de Pla). 3. La solution generale est sous la forme : - T(n) = Ara + Bri+ ... + M rom

*(*genéralement on ne dépasse pas forcé ou fe-3). *E*x*emple : E*quation de Fibonacci : . T (n) = şo

si nuo

si n=1 (1 \* TCR-A) + T(n-1) sin > A on ajoute (+4) pour chaque terme, on obtient.

si no quee s(n) =T(n)+1

si na (S(n-1) +5ln 2) si nx 1 Le palyname associé est : p(re) = .1 - 1

dant les racines sont r= 1. ve et rz = 1+ N5 La salutuen générale est donc : -

S() = A (4-5) " + B (1 + U5)

-> 5(0) = 1 1 AtBef et s(n) = 1 c A (^- S) + B (4 + 15) = A

DA - 54. N5 et B = 5 s(n) - (5 4.05 ) ( ^- NEJ" + (505)( ^ + NS)"

\*T() = 0 ((1 + Ws) aj

Rés*olutio*n de l*'équati*on de second se récumence av*e*c second membre - T(A) = x T(n-1)+ a2 T(n-2) + ... + KxT(n- k) + F Encemple éa de Fibonacci: T(n)= T(n-1)+ T(-a) + 4 - cas particulier: T (n) = an T (n=4) + f(n) Résolution:

T(n)= xn T(n-1) + f(n)

= xH (XA T(n-2)) + f(n) = x 2 T(n-2) + an T(n-1) + f(n) = x2 (anT(n-3) + f(n=2), an f(n-A) + f(n) = a T(n-3)\* + xk f(-2)+än fin-a) + (n)

= que ai T(0)+ ax^3 (1) + Gă f (2) + ... + xq f(-4)=f(n) T(a) = xh (T(0) + En

Exemple : Tour de Hanoi :

Ten) Eso

. Si no 12 T(n-1) + 4 si nyo on applique la formule

T(1) - TA -4) + (n) = < (T(0)+ $ ) $ T(n) = *2*" (O+ En

o T(m) = {- c'est un algorithme de comptencite exponentielle. Si on suppose que le déplacement d'un disque dure une

minute; pour résoudre un algorithme de 64 disques, il faut compten:

(a) = 2" - 1 min = 35096 ,5 milliards d'années.

Divisen

pour

regner

Principe :

Le paradigme "Divisen pour régnen" donne lieu à 3

étapes á chaque niveau de récunsivité : - Diviser le probleme en un centain nombre de sous

problemes. - Régner sur les sous - problemes en les résoluant

récunswerment, ou si la taille est assez rêduite, - Pe résoudre directement.

- Combinen les solutions des sous - problemes en une

solution complete da probleme initial. La sécurrence définissant le temps d'escécution d'un algorithme

"dwisen pour régnen" se décompese suvant les étapes des

paradigme de base: 1. Si fa taille du probleme est suffisamment réduite, nec poun une : centaine constante c, fa résolution est directe et consomme un

temps constant (1). a. Sinan, on dwise be probleme en a sous - problemes chacun de taille

Alb de la taille du probleme initial. le temps d'encécution total se décompose en 3 parties :

- D(n): le temps nécessaire ,a' fa dwision du problemes en s.p. - a tenib): le temps de résolution des a s.p. -Cin): le temps nécessavie pour construire la safutien finale.

Ten) = 0(1)

sinca a T(016) + D(n) + ((n) sinon

me de re*g*nen oma *te tep*roblem

-10 *E*xemple: Rechenche du marcimum d'un tableau :

int Maximum (int tabl], int feft, int right)

{ int Ra, ,mi

if (left = = right)

return fabčfeft] ; else

m = (heft + right) *1*2 ka = maximum (ne, left, m) k2 = marimum (s. mat, right) If (RA ) = R2) nghan Bá

si n=1

T(n)- Ş0(1) else

return ka?

*(*2T(n*/*a)+2 sinon

y un seul man

Excemple : Rechenche du manimum d'une fonction unimodale:.

int marf (int tab[], int feft, int right)

{ int m= (right +Pept)/2

if (tab (m] > tab [m +^] 82 tab [mm] > tab[m - 1])

return fab[m]; if (ab [m]> tab ['m + ]] = T(n) = T(1/2)+2

return manf (tab, left, m); Pene) return mare fltab, m+ a, sight); nase 1 (0-1) f(+1) . Condition d'arrêt:

\* mase > f (x - 1)

et mane > Pix+1)

**ronac**

no-1

no Mt1

Résolution des récurrences « Dwis*e*r pour régner"

- Theorème :

Soit T(n) = a Tanib) + f(n)

VOD()+ con) \* Pour calcupen la complexité, on doit comparerf (o) a n rog - car c'est la complensité de la resolution des a sous probe

\* (logo a). E est la bonne marcimale de T(n). 1.4 si aT(n lb) = f(n) alons Tra) = ln logb a logn) . Ho\* si alinib) < fini

- (dans ce cas f(i)= enega si f(n) = 2 ( logo a)+ Ej

et af(n/b) & a f(n) pour can

alons TCM) - Coin)) . 3- + si a Tanib) > Pln):

si f(n) = oln (logo a)-E) :

alons T(n) = a (n Pogba - Théorème di Dons qu'on peut ese primer f(n) sous forme

de polynâme (e xnx)]

T(n) = a Tilb) + f(n) cink \* si asbk :

T(n) = ln Pogbang Con campare à chaque

fois a = b\*

f(n) et a foglal et \*Si

on choisit la plus - T(o)= G (nk fogn) grande valean).

+ si a BK ::.. Inse. (2)

342

-12 *Ex*emple: M*ultip*lica*t*ion naive evee *de matric*es:

(multiplication de matrices.camées de taille n) algorithme naif:

Multiplier (A, B, C)

For iad to n de

For j = 1 ton de

Clip) = 0 For k=1 to n de

C(,j) = c(0,8) + A (1;&)\* BCRij) End for End Fon Endfor

os L'algo effectue End-Multiplier

O(n) multiplications. On décompose des matrices A, B et c en sous-mablices

de taille n/2 n/2. *L*'équation C= AB peut apons se récnine :

→ on obtient

1 = a + bg ; s= ag+bh ; ta ce + df ;u=29+ dh On peut done dériver un algo diviser pour régnen dont la complessité est donnée par la récurrence:

Tin) = 8 Tin*/*2) + O(n) 4 opérations &

Catinla)]-+ 2 multiplications de matrices canées de taille nie 1. et une addition O(n) Puisque a = 8 ; b=2 logo a = 3; f(n)= (22) = (12logia .cani (Pogba) - € = 3 -> E=..

\* L'afganithme a donc une campfencité en O(n3). Si on applique le qe theoreme:

f(n)= o(na) sous la forme cine aves .c=1, p = 2

a>bk (83 22) donc :.

T(A) = O(npogla)= a (n laga 8) = 3 (23)