Gestiunea memoriei (Storage Management)

Discutam despre modulcum este gestionata memoria alocata unui program pentru executia sa. Programatorul nu este preocupat de modul in care un limbaj paricular isi gestioneaza memoria pe care o are la dispozitie dar modul in care este implementat acest lucru , influenteaza foarte mult performantele programului (de exemplu, un lmbaj care nu are implementata alocarea dinamica nu permite recursivitatea).

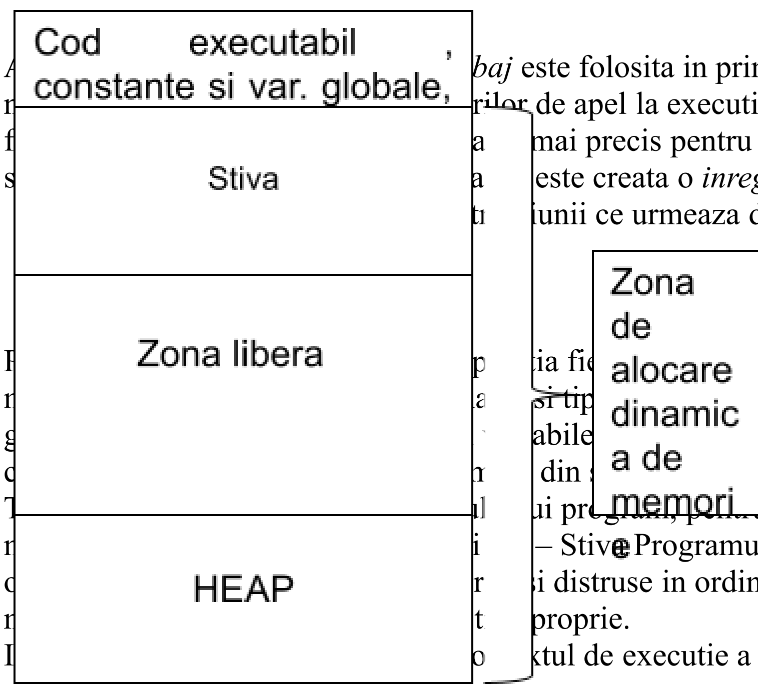
Alocarea statica de memorie : este facuta in timpul compilarii si nu este schimbata in timpul executiei programului. Daca o variabila este alocata static adresa de memorie din interiorul programului unde sunt memorate valorile respectivei variabile este fixata in faza de compilare a programului.
avantaje:

- executie eficienta (nu cere timp de executie pentru alocare)
- nu trebuie incluse rutine de alocare dinamica deci codul generat mai scurt.(nu este intodeauna adevarat)

Alocarea dinamica de memorie - se refera la alocarile de memorie efectuate in timpul executiei programului cu ajutorul unor rutine de management al memoriei. Exista doua tipuri :

- alocare controlata de limbaj
- alocare controlata de programator (cereri explicite). Eliberarea se face fie explicit fie automat (garbage collector)

Principial, harta memoriei alocata pentru executia unui program arata astfel:

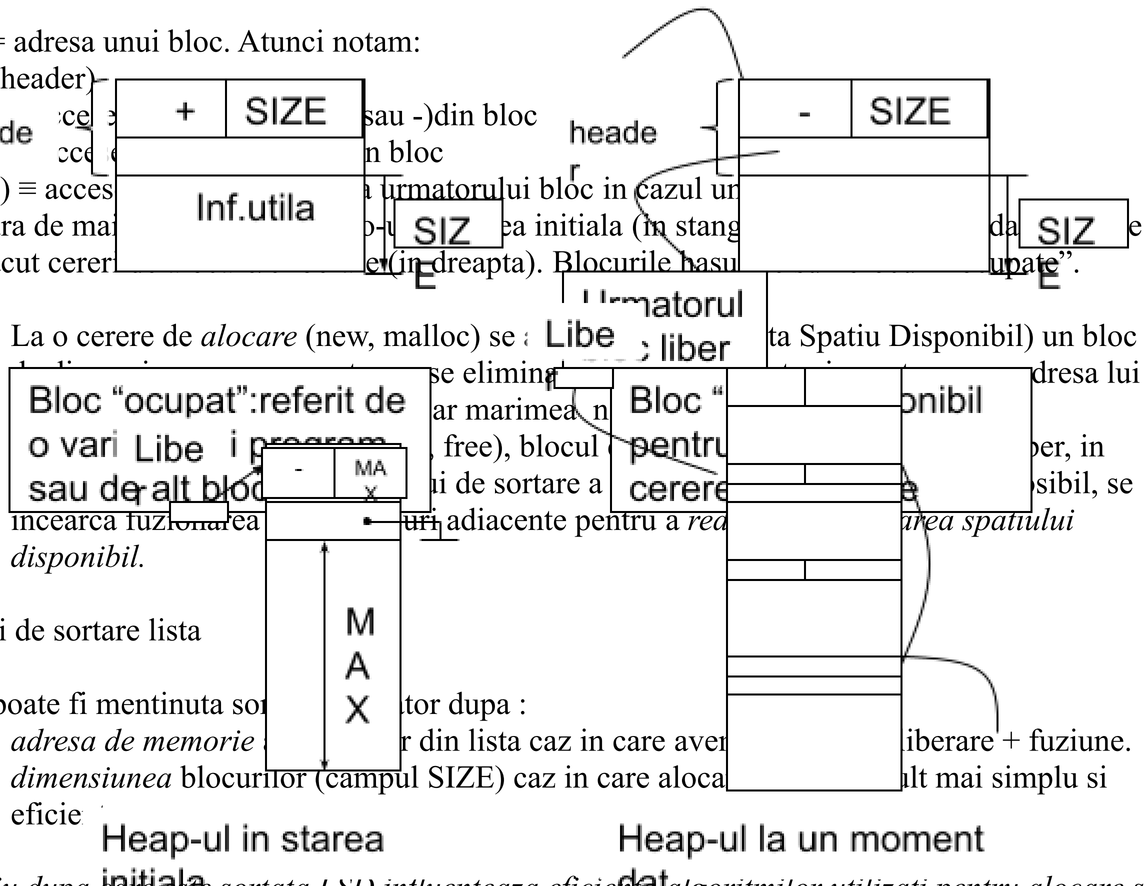


A[...] *baj* este folosita in principal pentru a asigura memoria r[...]rilor de apel la executia unei functii. Pentru aceasta se f[...]a [...]mai precis pentru fiecare thread). La fiecare activare de s[...]a [...]este creata o *inregistrare de activare* care contine : [...]iunii ce urmeaza dupa "call")

[...]ia fi[...]n inregistrarea de activare, din r[...]si tip[...]or si a variabilelor locale) putandu-se g[...]abile[...]terminarea procedurii, inregistrarea c[...] din [...] T[...]ui pr[...] toate procedurile active la un r[...] – Stiva Programului – si unde sunt depuse secvential in c[...]i distruse in ordinea inversa(lista LIF∅). In sistemele cu r[...]proprie. I[...]xtul de executie a procedurii pentru care a fost creata.

Alocarea dinamica *controlata de utilizator* este in limbajele care permit descrierea de structuri dinamice : Java, C, C++, C#, Ada, Modula, LISP. Majoritatea limbajelor au aceasta facilitate.

Zona de memorie din care se fac alocarile si eliberarile (in ordine arbitrara) este cunoscuta sub numele de HEAP.
Pentru incepu facem urmatoarele presupuneri:
– spatiul de memorie poate fi alocat in blocuri de lungime variabila.
– atat alocarea cat si eliberarea se fac explicit(new, dispose sau malloc, free, etc.)
– sistemul de gestiune a spatiului de memorie mentine o lista a blocurior libere de unde se aloca blocurile. Initial aceasta este constituita dintr-un singur bloc de dimensiunea maxima disponibila
– blocurile au urmatoarea structura :



Fie b = adresa unui bloc. Atunci notam:
h = lg(header)
T sau -)din bloc
S n bloc
Urm(b) ≡ acces urmatorului bloc in cazul un
In figura de mai ea initiala (in stang da e
s-au facut cereri (in dreapta). Blocurile hasu upate".

- La o cerere de *alocare* (new, malloc) se ta Spatiu Disponibil) un bloc se elimina dresa lui ar marimea n
- free), blocul er, in i de sortare a sibil, se incearca fuziunea uri adiacente pentru a *rea area spatiului disponibil.*

Criterii de sortare lista

LSD poate fi mentinuta so tor dupa :
- *adresa de memorie* r din lista caz in care aver liberare + fuziune.
- *dimensiunea* blocurilor (campul SIZE) caz in care aloca lt mai simplu si eficie

Criteriu dupa care este sortata LSD influenteaza eficienta algoritmilor utilizati pentru alocare si eliberare.

Cei mai cunoscuti algoritmi utilizati:
FirstFit → alege primul bloc din LSD la care size(bloc) >= lungimea ceruta.
BestFit → alege cel mai mic bloc care este mai mare decat dimensiunea ceruta.
i.e. daca se cere un bloc de dimensiune = n ,se alege blocul i a.i. :

$$size(i) - n = \min_{j} [size(j) - n]$$

Algoritmul BestFit de alocare

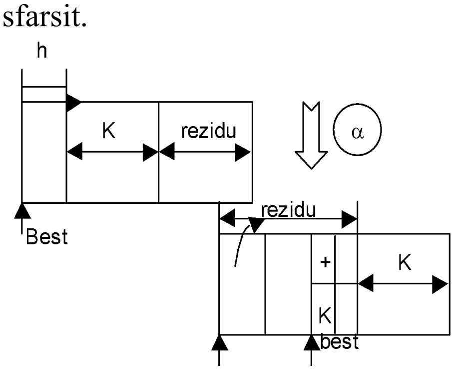
a) Cazul in care LSD este sortata in ordinea *crescatoare a adreselor* blocurilor.

EMBED Word.Picture.8

- Procedura de <u>alocare</u>, primeste lista spatiului disponibil (i.e. pointer la primul bloc din lista) si dimensiunea K ce se cere sa fie alocata.
Se cauta in LSD blocul cu dimensiunea cea mai apropiata de K (dar mai >= decat K).
Daca se gaseste un astfel de bloc, se intoarce adresa acestui bloc si in plus se modifica LSD prin eliminarea acestui bloc din lista blocurilor libere.
Daca nu pot face alocare ⇒ intorc NIL.

procedura BestFit(intrare :Liber,k; iesire :Liber,adresa) este
    / * variabile folosite (semnificatia lor) :
        best : contine adresa "celui mai bun bloc" gasit pana la pasul curent
    bloc : folosita pentru a parcurge LSD din bloc in bloc
        rezidu : folosita pentru memorarea valorii :size(best) - k a blocului
    "best" * /
atribuire bloc ← Liber / * adr. 1^{st} bloc din lista * /
    rezidu ←MaxInt;   //OBS : h = lg (header)
    best ← NIL;
cat timp (bloc ≠ NIL) repeta
    daca (size (bloc) >=k) and [size (bloc) – k < rezidu]
        atunci best ←bloc;
            rezidu ← size(bloc) – k;

    bloc ← URM(bloc);
daca best = NIL atunci adresa ←NIL
altfel daca (rezidu <h) atunci Liber ← Delete (best,Liber);
    adresa ←best;
altfel size(best) ←rezidu – h; vezi desenul notat "α" mai jos
best ← best + rezidu;
    size(best) ← k;
    Tag(best) ← '+';
    adresa ← best;
sfarsit.



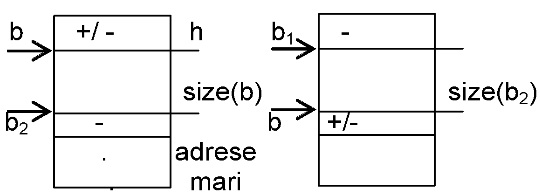
Procedura se poate imbunatati pentru cazul cand (rezidu – h) este foarte mic

b)Daca lista Liber, este sortata crescator functie de lg blocurilor atunci:
cat timp (bloc ≠ NLI) and (size (bloc) <k) executa
    bloc ← urm(bloc);
return (bloc) ; / * intorc adr. blocului * /

Eliberarea

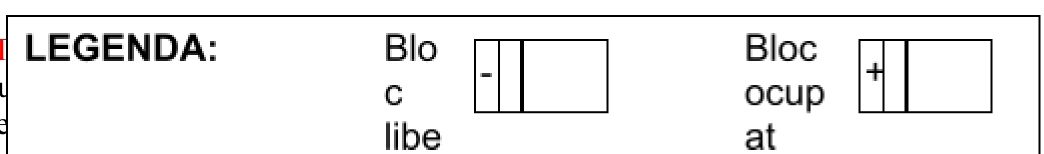
Presupunem Liber – sortata crescator dupa adresele blocurilor din lista.
Fie b – blocul de eliberat.
Fie pred, succ blocuri din lista Liber a.i. adr(pred) < adr(b) < adr(succ)

Daca : adr (succ) = adr(b) + lg(header) + size(b)
    <u>atunci</u> putem fuziona cele doua blocuri ⇒ fuzionare in amonte
Daca : adr(pred) = adr(b) – size(pred) – h ⇒fuzionare in aval
Daca se verifica ambele conditi va avea loc o fuzionare dubla.
Mai jos: b1=pred si b2=succ

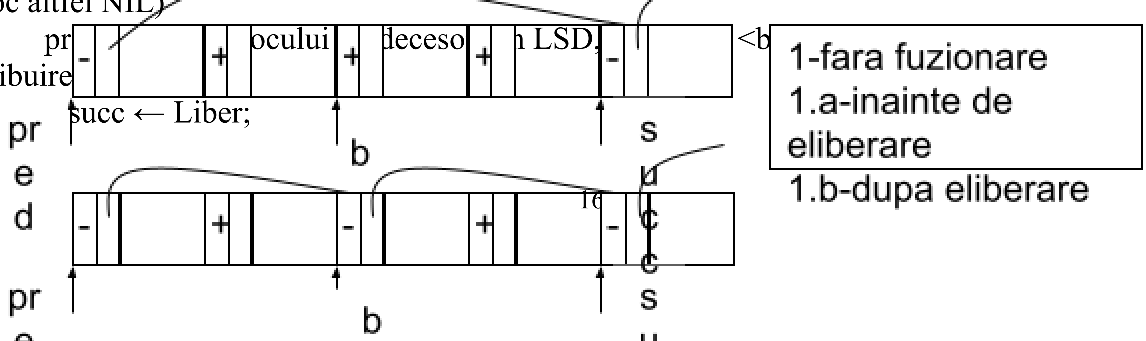


Cateva situatii posibile sunt prezentate in diagramele de mai jos :
- in figura 1 este cazul de eliberare a blocului b dar nu se poate face fuziune
- in fig. 2 se poate face fuziune in amonte
- in fig. 3 se prezinta cazul de fuziune dubla

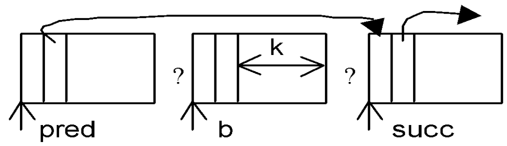
ATENT
procedu
/ * Libe
b – adresa blocului care se elibereaza



succ = adresa primului bloc din LSD care se afla la o adresa din b (daca exista acest bloc altfel NIL)
pr ocului deceso LSD <b
atribuire
succ ← Liber;

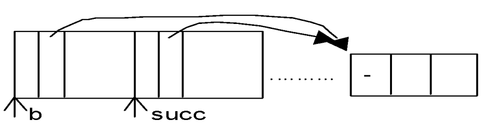


pred ← NIL
cat timp (succ ≠ NIL) and (succ <b) repeta
atribuire pred ← succ
succ ← URM(succ)



/ * in acest moment avem :
⇒Testez conditiile de fuziune * /

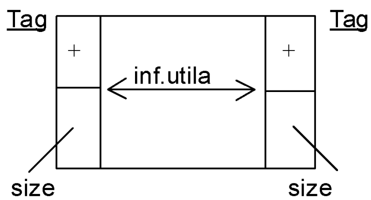
daca succ ≠NIL / * i.e. (∃) in LSD un bloc la o adresa > b * /
atunci daca (b+h+k = succ) / * conditia fuziune "amonte" * /
atunci size(b) ← k + size (succ) + h;
urm(b) ← urm(succ);
i.e / *

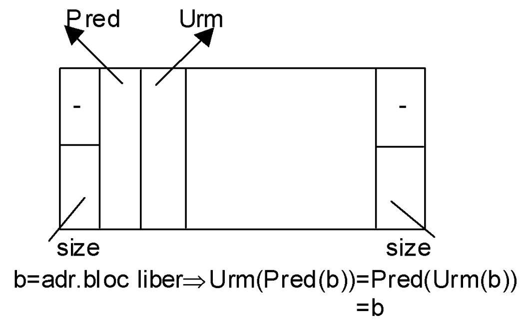


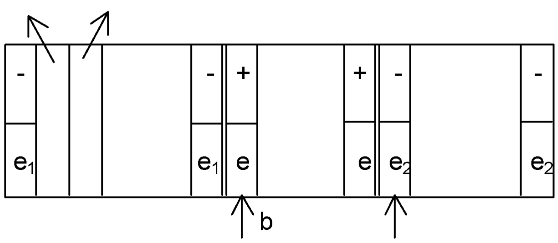
/ *
altfel urm(b) ← succ
altfel urm(b) ←NIL / * nu exista in LSD alt bloc dupa b * /
daca pred ≠ NIL / * i.e. (∃) bloc in LSD la o adresa < b * /
atunci daca pred = b – size(pred) ← size(pred) – h / *conduce la fuziune in "aval" * /
atunci size (pred) ← size(pred) + size(b) + h;
urm(pred) ← urm(b);
altfel urm(pred) ← b;
altfel Liber ← b;
sfarsit.

OBS : → La alocare preferam ca lista LSD sa fie pastrata sortata crescator in functie de lungimea blocurilor din lista .
→ La eliberare in schimb, daca blocurile din lista apar in ordinea adreselor lor ⇒ testez mai usor adresele de fuziune.

Daca aleg alta structura pentru blocuri , pot sa pastrez lista sortata dupa lungime si pot sa testez comod si conditia de fuziune.





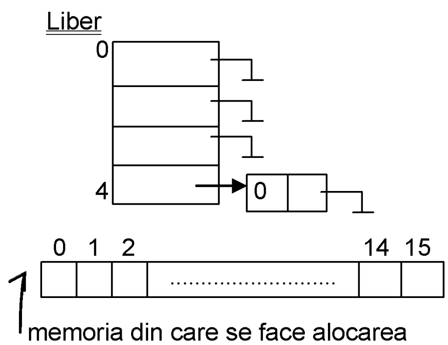


Testez conditia de fuziune : a) amonte Tag(b + 2h + size(b)) = ‘-’
aval Tag (b-1) = ‘-’
OBS : - Ca urmare a fuzionarii este posibil sa fie nencesara sortarea listei LSD .
→ La oricare din metodele de pana acum apare un dezavantaj : fragmentarea spatiului de memorie.

Buddy System (Sistemul cu “camarazi” (Knuth))
-
Principiul metodei :
→ Cu aceasta metoda se aloca blocuri de memorie a caror lungime trebuie sa fie o putere a lui 2 (i.e.2,2’,2^2,...)
Daca se afce o cerere de alocare cu o lungime $\neq 2^k$ $\Rightarrow$ se aloca primul bloc acoperitor ca lungime si care are o lungime = cu o putere a lui 2.
→Blocul de memorie din care se fac alocarile, este presupus de dimensiune 2^M .
→Evidenta blocurilor libere, se tine cu ajutorul unui vector de liste.
→Fiecare componenta i a acestui vector mentine lista a blocurilor libere de dimensiune 2^i sau lista vida daca in momentul considerat nu exista nici un bloc liber de dimensiune 2^i .→Initial ($\exists$) un singur bloc,avand dimensiunea intregii memorii disponibile 2^M .
→ La o cerere de alocare a unui bloc de dimensiune 2^k ,se cauta un bloc liber de aceasta dimensiune.

daca exista $\Rightarrow$ se aloca acest bloc
daca nu exista ,dar ($\exists$)un bloc de dimensiune mai mare, se incepe un proces de injumatatire, plecand de la acest bloc, proces care se continua pana se obtine un bloc de dimensiune 2^k .
OBS : Cand un bloc este divizat in doua , cele doua jumatati poarta numele de “camarazi” (“muguri”).

EX : M = 4



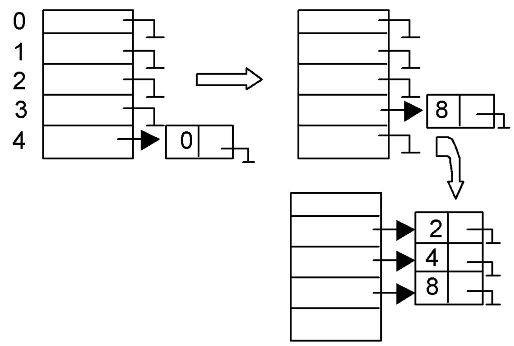
!!!!!!!!!!!!

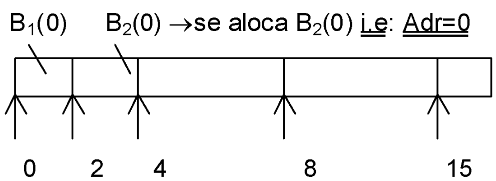
OBS : $\rightarrow$ O intrare i din vectorul Liber (i.e. Liber [i]), este un pointer spre lista blocurilor libere de dimensiune 2^i .

$\rightarrow$Fiecare element al listei contine in primul camp adresa blocului (adresa de inceput).

OBS : $\rightarrow$Cand un bloc de dimensiune 2^{k+1} se injumatateste rezulta doua blocuri (“camarazi”, “muguri”) pe care le notam B_k (A) si B_k (A’) care au propietatile :
fiecare bloc are dimensiune = 2^k
A mod $2^k = 0$ si A’ mod $2^k = 0$ (i.e. adresa la care incepe oricare din cele doua blocuri (A sau A’) este multiplu de 2^k.

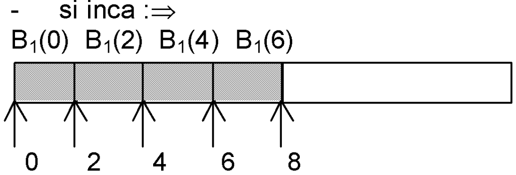
OBS : $\left\{\begin{array}{l} K = \text{ordinul blocului} \\ A = \text{adresa blocului} \end{array}\right\} \Rightarrow B_k(A) \rightarrow$ bloc de ordinul K si adresa A.
Fie in exemplul nostru o cerere de alocare a unui bloc de dimensiune = 2 cuvinte ; i.e. cerere de determinare B_1(Adr) si vreau sa aflu adresa Adr la care se face alocarea: $\Rightarrow$ avem succesiv:
!!!!!!!



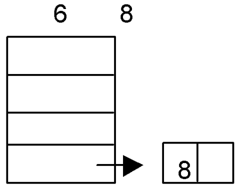


OBS : Injumatatirea se continua cu "camaradul"
din stanga pana la 2^k ,iar cel din dreapta e introdus ca bloc liber.
Acum presupun o noua cerere $B_1(Adr)$:

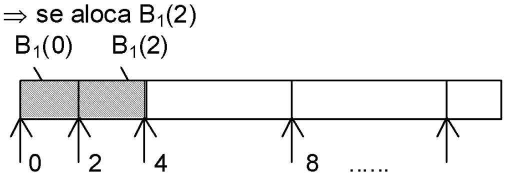
- si inca :⇒



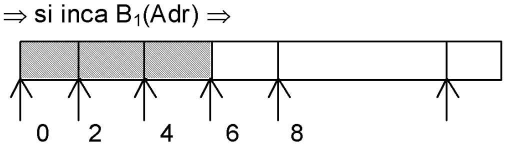
si Liber →



⇒ se aloca $B_1(2)$

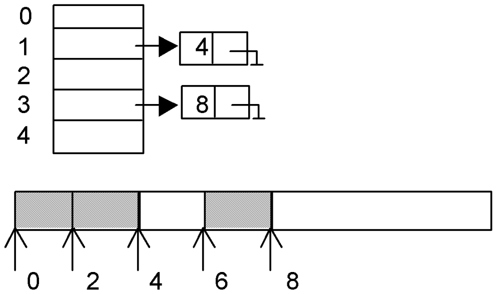


⇒ si inca $B_1(Adr)$ ⇒

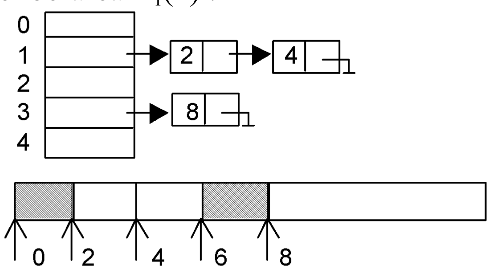


Eliberarea

eliberarea blocului $B_1(4)$:



eliberarea $B_1(2)$:



OBS : →pentru a respecta algoritmul nu pot sa fuzioneze doua blocuri vecine,decat daca ele sunt "camarazi" i.e. : au provenit din divizarea aceluiasi bloc (la noi , $B_1(2)$ si $B_1(4)$ nu sunt "camarazi" intre ei).

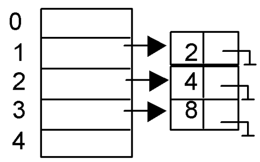
DEF : Fie $B_i(A)$ si $B_j(A')$ doua blocuri eliberate.Conditiile ca cele doua blocuri sa poata fuziona : sa aiba aceeasi dimensiune ⇒ $i = j$

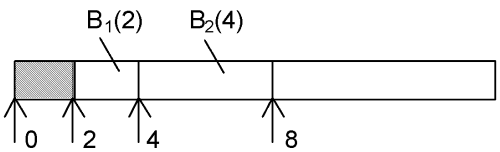
$|A – A'| = 2^i$ (blocuri adiacentede dimensiune 2^i)

c) $\min(A,A')$ nod $2^{i+1} = 0$ i

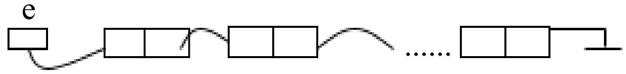
$2^{i+1} = 2*2^i$

adresa blocului care ar rezulta dupa fuziune (!!sa fie la multiplu 2^{i+1})
→La noi in ultimul caz : min (A,A') = 2 ; 2 nod 4 ≠ 0 !!
· Si acum o noua eliberare : $B_1(6)$.
Se observa ca $B_1(6)$ si $B_1(4)$ indeplineste conditiile de fuziune:
acceasi dimensiune
$|6 - 4| = 2 = 2^1$
min(A,A') = min (4,6) = 4 si 4 nod $2^{1+1} = 0$

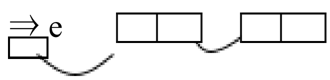




Pentru evidenta spatiului disponibil ,am o variabila TabLis : array [0..M] of lista (tablou de lista) unde 2^M = dimensiunea totala a spatiuluidin care fac alocarile.
Presupun ca pentru Lista , am definiti urmatorii operatori: First , Insert , Delete.
Ex: e



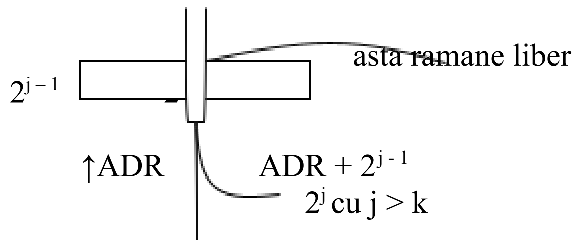
atunci : First (e) → a1
e → Insert (a_0 ,e) ⇒ e



e → Delete (a_i,e) ⇒ elimina a_i din e.
Presupun ca se afc cereri de alocare B_k (A) i.e. se cere alocarea unui bloc de dimensiune = 2^k.Daca pot sa fac alocarea intorc in A adresa blocului alocat, modificand corespunzator TabLis.Daca nu se poate face alocarea ,intorc NIL.

```
procedura Alocare (TabLis , M.k ; TabLis,Adr) este j←k;
        cat timp (j<=M) and (TabLis[j] = NIL) executa
j←j + 1;
daca j > M
        atunci ADR ← NIL  / * nu mai exista spatiu */
        altfel    ADR ← First(TabLis[j]) ; / * ADR aici fac alocare * /
                TabLis[j]←Delete (ADR,TabLis[j]);
                cat timp(j <k) executa
                        TabLis[j –1]←Insert(ADR + 2^(i-1) ,TabLis[j -1]
```

/ *



*/

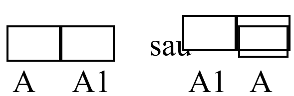
j ← j – 1 ;

sfarsit.

OBS : First $\begin{Bmatrix} \text{Car} \\ \text{Top} \end{Bmatrix}$ Insert $\begin{Bmatrix} \text{Push} \\ \text{Cons} \end{Bmatrix}$ Delete $\begin{Bmatrix} \text{Cdr} \\ \text{Pop} \end{Bmatrix}$

procedura Eliberare (TabLis,M,k,A) este
j ← k;
gata ←Fals;
cat timp (j <=M) and (not gata) executa
daca ○ exista A1 ∈ TabLis[j]care poate fuziona cu A i.e.: $|A – A1| = 2^j$ and min (A,A1) nod $2^{j+1} \equiv 0$
atunci / * A1 ≠NIL * /
TabLis[j] ← Delete(a1,TabLis[j])
A ← min(A,A1);

/ *



* /

j ← j = 1 ; / * ⇒ un bloc liber de dimensiune dubla care poate mai fuzioneaza si el * /
altfel TabLis [j] ← Insert(A,TabLis [j])
gata ← Adevarat;

sfarsit.