BT20CSE112 – Kaustubh Shivshankar Shejole

25702

Concepts In Programming Languages – Assignment 3

Heap Management :

Program to implement heap management scheme implementing ‘allocate’ and ‘deallocate’ functions ensuring that adjacent free blocks are merged together.

Heap management scheme used : Fibonacci heap dynamic pool adjustment.

To reduce the cost of allocation separate free lists for blocks of different sizes which are Fibonacci numbers is maintained. Each request is rounded up to the next standard size. Here dynamic division would be used which means that smallest possible block will be given for that request.

If we only have a block greater than given request then that block is divided into 2 parts : Let set that block was of size Fn. Then it would be divided into Fn-1 and Fn-2 and then it would be decided which block to give it may happen that nobody is able to satisfy then complete block would be given:

Algorithm for allocation:

Request is checked first if it is outside the constraints error is printed and here we simulate NULL as -1 value and it is returned. Otherwise the address of allocated block is returned.

Demand means the rounded value of request.

An index is calculated from that demand.

Then if free\_list\_array[index] is empty: -> then next index is searched

Else that block in index is given and block is removed from free\_list.

If demand is equal to block size then give that block

Else if demand is equal to previous Fibonacci block size -> give that

Else demand is equal to second previous Fibonacci block size -> give it

Else: do the same process for second previous

* And the constraints of index also should be considered.

If not found biggest block of demand:

Print error message that NULL returned

* Negative and zero requests not allowed.

In this program arrange function does the work of searching and arranging:

The above algorithm is implemented there with some corner conditions like if we have block of 2 and we have demand 1 so we have to give 1 and store other in same index-1 position and it is free. For others index-2 and index-1 positions are used.

In doing Algorithm free\_list should be updated as needed.

And lastly filled\_list also.

Requirement : List of free lists of different standard size blocks:

Fibo\_list: list of Fibonacci numbers upto our requirement

Free\_list\_array: list of list of free\_address with each\_list having standard

Size specified by index as size=fibo\_list[index].

Filled\_list\_array: same as that of free\_list\_array only difference is that it stores filled address.

In our case Fibonacci Numbers would start from 1 as 1 , 2 , 3 , 5 , 8 just for making it adjustable with our free\_list block size and also with filled\_list block size.

Deallocation:

Requirement: after deallocating from filled\_list we would have to biggest Fibonacci block that could be made by merging and checking address previous and next to block so that in future if there is a bigger request we could satisfy that.

Algorithm for Deallocation:

Take the address of block to be removed

Search it in filled\_list\_array because here we don’t know the size of that block

If not found: print Error message

If found store the index:

Remove address from filled\_list\_array[index]

Check that prev\_address is free or not(not applicable to index 0):

If(free) ->

Append address to free\_list\_array[index]

address = prev\_address

Because from this address the bigger block will start now

Remove prev\_address from free\_list\_array[index-1]

Merge the next blocks if available using merge blocks function.

Else If(index is 0 and next\_address is present in free\_list\_array[0])

Remove next\_address from free\_list\_array[0]

Again call merge next block function for index+1 now

Else If(index is 0 and previous address is present in free\_list\_array[0])

Remove previous address from free\_list\_array[0]

Address = previous address

Call merge blocks function for index+1 or(simply 1 as index is 0) now

Else

Call merge blocks function for index

What does this merge function do?

If found next address at next block it merges them and again calls merge\_blocks(index+1,address)

Also checks for previous address if it is free and does their merging too.

Else appends address in free\_list\_array[index]

if previous address is present in free\_list\_array[index-1]

if( Not found address in free\_list[index])

{

free\_list\_array[index].append(address);

address = previous address

remove previous address from free\_list\_array[index-1]

index = index -1

}

If(index+1 less than size of free\_list\_array and

We found next\_address in free\_list\_array[index+1]):

Remove next address from index

Call merge next blocks for index+2 if constraints followed

Else :

Append address at free\_list\_array[index]

Important Note:

We get previous address by subtracting fibo\_list[index-1] from our address of block at index index. For next address we add fibo\_list[index] to our address.

So in this way deallocation and merging happens more details will be there in our code.

Like var = malloc(21) for allocation request of 21 and it will store the address of

The block allocated in var

free(var) will do deallocation of that memory and merging will also happens so as to have a large enough block for next request.

Heap size is intentionally not predefined but this can also be done. The intention

is to give the size required manually according to user’s demand.

So this was in short about the whole program.

List of Defined Functions:

**int near\_bigfib(int n**) : gives fibonacci number greater than equal to n and populates fibo\_list also.

**int near\_fib(int n,string sign=”>=”)** : gives nearest fibonacci number greater than equal to our request with '>' sign gives bigger one with '<' sign gives smaller  one  
**void populate\_free\_list\_array()**: fills free list array  
**void populate\_filled\_list\_array()**: fills filled list array  
**int allocate(int r)** : allocates given request  
**int allocate\_helper(int demand,int index,bool repeated=false)** : useful in allocation

**int satisfy\_demand(int demand,int index)**: appends address at filled\_list\_array[index] and removes address from free\_list\_array[index].  
**int arrange(int demand,int index)** : find smallest block possible allocates it and returns starting address of block.  
**bool search\_in\_list(int address,list l)** : searches a given address in simple list and returns true if found otherwise returns false.  
**void deallocate(int j)** : deallocates block with address j  
**void merge\_blocks(int index,int address)** : if found a next address merges it and check for later as well otherwise appends that address at that index.

**int malloc(int request)** : allocates request memory on heap and returns starting address of the block allocated.

**void free(int var)** : frees up the memory pointed by var which has address of the block to be freed.

Thank you Sir