2024.1

Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Departamento de Informática
Curso de Ciência da Computação - Disciplina: Estrutura de Dados

Segunda Prova.

Obrigatório utilizar os tipos de dados indicados nos protótipos das funções

1. Escreva um algoritmo que recebe uma lista linear simplesmente encadeada e move o último elemento da lista n posições pra frente. Não pode alocar novos nós da lista. Se a lista tiver menos que n nós coloca o último na primeira posição da lista.
   int MoveNPosicoesPraFrente (Sllist *l, int n)

2. Faca um algoritmo que recebe duas listas circulares duplamente encadeadas (L1 e L2) inclui todos os nós de L2 em L1, de maneira intercalada. Não pode alocar novos nós.
   DLList *Intercala(DLList *l1, DLList *l1,)

3. Escreva um algoritmo Incomuns (L1, L2) , que deve retornar um valor inteiro igual ao número de valores que estão em L1 e não estão em L2. L1 e L2 são circular simplesmente encadeadas.
   int Incomuns (Sllist * l1, Sllist l2, int (*cmp) (void *, void *));
   cmp retorna 0 (zero) se os dois argumentos forem iguais.

4. Escreva um algoritmo que recebe uma lista circular duplamente encadeada L e remove um elemento especificado pela chave Key, juntamente com seu vizinho anterior (prev) se ele existir.
   int RemoveEspecificadoEAnterior (DLList *l, void *key,
   int (*cmp)(void*, void*))

60/100

```
1) int moveNpositionsprefrente(SLlist* l, int n){
    if(l!= null && n>0){
        if(l->first != null){
            sllnode* prev = null;    sllnode* prev2 = null;
            sllnode* curr = l->first;
            while(curr->next!= null){
                prev=curr;
                curr=curr->next;
            } int i=0; prev->next = null;
            if(sllnumnodes(l) >=n){
                [illegible] prev 2 = l->first;
                for(i=0; i<n; i++){
                    [illegible]
                    while(prev2->next!= prev){
                        prev2=prev2->next;
                    }
                    prev= prev2;
                }
                next= prev->next;
                prev->next = curr;
                curr->next = next;
            } else {
                prev= l->first;
                l->first=curr;
                curr->next= prev;
            }
            return true;
        }
    }
    return false;
}
```

cadê? ? ??

20/30

dll

2) Dllist* intercala (Dllist* l1, Dllist* l2){
if(l1!= null && l2!=null){
if(l2->first!= null){
Dllnode* cur1 = l1->first;
Dllnode* cur2 = l2->first; Dllnode *prev;
if(cur1 == null){
l1->first = cur2; cur2=cur2->next;
While(cur2!= l2->first){
prev = cur2->prev;
prev->next = cur2;
cur2 = cur2->next;
}
return l1;
} else {
Dllnode * next = cur1->next;
cur1= cur1->next; cur2 = cur2->next;
~~cur1!=l1->first && cur2!= l2->first~~
While(~~cur1!= null && cur2!= null~~){
cur1->next = cur2; - grebous este!
cur2->prev = cur1;
cur2->next = next;
cur1 = next;
next = cur1->next;
cur2 = cur2->next;
}

if(cur1 == l1->first && cur2!= l2->first){
prev = cur1->prev;
prev->next = cur2;
While(cur2!= l2->first){
prev = cur2;
cur2->prev = prev->prev; // pleca a anterior de cur2 que é igual a prev //
cur2 = cur2->next;
prev->next = cur2;
}

return l1;
}
return null;
}

cmp=0 → uguali

3. int incomuni (Slist *l1, Slist *l2, int (*cmp)(void*, void*)){
if(l1!= NULL && l2!= NULL){
if(l1->first!= NULL && l2->first!= NULL){
Nodo* cur1 = l1->first->next; prev1 = l1->first;
Nodo * cur2 = l2->first->next;
Nodo* prev2 = l2->first;
int incomuni=0; int stat = cmp(cur1->data, cur2->data);
While(cur1!= l1->first){
While(cur2!= l2->first){
~~if(cmp(cur1->data, cur2->data)!=0){~~
~~incomuni++~~
if(stat==0){ cur2 = cur2->next; stat=1;
Break;
} else{
cur2 = cur2->next;
stat = cmp(cur1->data, cur2->data);
}
}
cur1 = cur1->next;
if(stat=1){
incomuni++;
}
stat = cmp(cur1->data, cur2->data);
}
return incomuni;
}
}
return -1;
}

30/30

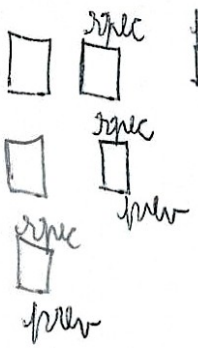
```
4. int removeespecialanterior( Dlist* l, void* key, int(*cmp)(void*, void*)){
    if( l != null && key != null){
        if(l->first != null){
            dllnode* prev = l->first;
            dllnode* cur = prev->next;
            int stat = cmp( cur->data, key);
            while( cur != l->first && stat != true){
                prev = cur;
                cur = cur->next;
                stat = cmp( cur->data, key);
            }
            if( stat == true){
                prev = cur->prev->prev;    dllnode * remove = null;
                if( spec->next == prev){
                    prev->next = prev;
                    prev->prev = prev;
                    remove = cur->prev;
                    free( remove);
                    free( cur);
                    l->first = prev;
                } else if( spec == prev){
                    ~~if( l->first == spec){~~
                    l->first = null;
                    remove = cur->prev;
                    free(remove);
                    free(cur);
                } [illegible] else {
                    prev->next = [illegible] cur->next;
                    cur->next->prev = prev;
                    if( l->first == [illegible] || l->first == cur->prev){
                        l->first = cur->next;
                    }
                    remove = cur->prev;
                    free( remove);
                    free( cur);
                }
                return true;
            }
        }
    }
    return false;
}
```

//algumas linhas foram pra estar fora do if/else, mas como não posso apagar, coloquei dessa forma//

30/30

4

→ normal → ultimo.



rplc → nlc == fplv;

elimina rplc è o ultimo

rplc == fplv;

elimina rplc è o ultimo diversa lista vuota

rplc == fplv && l → first == rplc.

elimina rplc è diverso.