פתרון

1.

פתרון של שאלת ADT

א.

struct RACS;

typdef struct RACS \*pRACS;

pRACS RACS\_Create();

int RACS\_AddMotion(pRACS con, ComplexMotion func);

Result RACS\_RemoveMotion(pRACS con, int index);

int RACS\_AddProgram(pRACS con, int\* prog\_arr, int num);

Result RACS\_RemoveProgram(pRACS con, int index);

Result RACS\_OperateProgram(pRACS con, Coordinate\* out, int index, Coordinate in);

void RACS\_Destroy(pRACS con);

struct RACS{

pList motion\_list;

pList prog\_list;

};

typedef struct{

int index;

ComplexMotion func;

} motion\_st, \*pmotion;

typedef struct{

int index;

int num\_funcs;

int\* prog\_arr;

} prog\_st, \*pprog;

int RACS\_AddMotion(pRACS con, ComplexMotion func)

{

pmotion element,new\_elem;

if((new\_elem = malloc(sizeof(motion\_st)))==NULL)

return -1;

new\_elem->func = func;

int max=0;

ListSetFirst(con->motion\_list);

elem = ListGetCurrent(con->motion\_list);

while (elem)

{

if (max<((pmotion)elem)->index)

max=((pmotion)elem)->index;

ListSetNext(con->motion\_list);

elem = ListGetCurrent(con->motion\_list);

}

new\_elem->index=max+1;

elem = ListInsertCurrent(con->motion\_list,new\_elem)

if (elem == NULL)

{

free(new\_elem);

return -1;

}

return new\_elem->index;

}

ג.

Result RACS\_OperateProgram(pRACS con, Coordinate\* out, int index, Coordinate in)

{

pprog pelem;

pmotion melem;

/\* Find program \*/

ListSetFirst(con->prog\_list);

pelem = ListGetCurrent(con->prog\_list);

while (pelem)

{

if (((pprog)pelem)->index == index)

break;

ListSetNext(con->prog\_list);

pelem = ListGetCurrent(con->prog\_list);

}

// Program not found

if (!pelem)

return fail;

/\* Find all the motions \*/

int size = ((pprog)pelem)->num\_funcs;

int\* prog\_arr = ((pprog)pelem)->prog\_arr;

ComplexMotion\* motions =

malloc(sizeof(ComplexMotion)\*size);

for (int i=0;i<size;i++)

{

melem = FindMotion(con->motion\_list,prog\_arr[i]);

// Motion not found

if (!melem)

{

free(motions);

return fail;

}

motions[i]= ((pmotion)melem)->func;

}

/\* Execute the motions \*/

for (int i=0;i<size;i++)

{

if (motions[i](out,in)==fail)

{

free(motions);

return fail;

}

in=\*out;

}

free(motions);

return success;

}

ד.

void RACS\_Destroy(pRACS con)

{

pprog pelem;

pmotion melem;

ListSetFirst(con->prog\_list);

pelem = (pprog)ListGetCurrent(con->prog\_list);

while(curr\_p)

{

ListRemoveCurrent(con->prog\_list);

free(pelem ->prog\_arr);

free(pelem);

ListSetFirst(con->prog\_list);

pelem = (pprog)ListGetCurrent(con->prog\_list);

}

ListSetFirst(con->motion\_list);

melem = (pmotion)ListGetFirst(con->motion\_list);

while(melem != NULL)

{

ListRemoveCurrent(con->motion\_list);

free(curr\_f);

melem = (pmotion)ListGetFirst(

con->motion\_list);

}

ListDestroy(con->prog\_list);

ListDestroy(con->motion\_list);

}

2.

1. משתמשים בשתי הערימות בנפרד, כאשר בהכנסת איבר מכניסים אותו לשתי הערימות.

יצירת המבנה כולל יצירת 2 ערימות: סיבוכיות .

הוספת איבר: מוסיפים לשתי ערימות, בסיבוכיות .

מציאת האיבר המינימלי והמקסימלי בסיבוכיות קבועה, ע"י שימוש בשתי הערימות.

הסרת האיבר המינימלי: הסרת איבר המינימלי מערימת מינימום ב-, ואילו הסרת איבר המינימלי מערימת מקסימום תהיה ב- כי יש קודם למצוא אותו בערימה. לכן סה"כ הסיבוכיות של פעולה זו היא .

הסרת האיבר המקסימלי:  באופן דומה.

1. המפתח המינימלי הוא כמובן השורש של העץ, ואילו המפתח המקסימלי הוא אחד הבנים של השורש (המקסימלי מביניהם). הכל בסיבוכיות קבועה.
2. האלגוריתם דומה לאלגוריתם  המקורי, רק כאן לא מספיק לבדוק את הבנים של הצומת, אלא גם את הנכדים. להלן:

* נתבונן בקבוצת כל הבנים (הנמצאים ברמה זוגית) והנכדים (הנמצאים ברמה האי-זוגית) של הצומת . מתוך קבוצה זו נבחר את הצומת בעל המפתח ה**קטן** ביותר. נסמן צומת זה ב-.
* אם  הינו **הבן** של , כלומר הוא נמצא ברמה הזוגית, אזי
  + אם המפתח של  קטן ממפתח של , החלף בין  ל-. (**כאן אין צורך בקריאה רקורסיבית כי החלפנו ברמה זוגית לאיבר בעל מפתח גדול יותר, לכן לא יכולנו לפגוע בתנאי  שם)**.
* אם  הינו **הנכד** של , כלומר הוא נמצא ברמה האי-הזוגית, אזי
  + אם המפתח של  קטן ממפתח של :
    - החלף בין  ל-.
    - יש לבדוק כעת שלא פגענו בתנאי **** של האב של . כלומר אם לאחר ההחלפה הנ"ל מקבלים כי המפתח של  גדול המפתח של ה**אב** של , יש להחליף ביניהם.
    - כעת יש לבצע קריאה רקורסבית של  על , על מנת לדאוג כי לאחר ההחלפה גם  מקיים את תנאי .

סיבוכיות הזמן של פעולה זו היא , כי יש כאן רקורסיה לעומקו של העץ.

1. האלגוריתם  זהה לזה של ,בהחלפת סימני השוואה, ובהתייחס אל  כאל בן או נכד עם המפתח הכי גדול.
2. האלגוריתם פשוט:

* אם  נמצא ברמה האי-זוגית, הפעל .
* אם  נמצא ברמה הזוגית, הפעל .

1. האלגוריתם זהה לזה של ערימה מקורית: מוציאים את האיבר המקסימלי (ראה ב'), ושמים במקומו את האיבר האחרון בעץ. כעת מפעילים  על צומת זה. הסיבוכיות  כסיבוכיות אלגוריתם . באופן דומה – הסרת האיבר המינימלי.

3.

3

2

1

c=0

a=2

a=1

a=0

ב.

--1--

B::B()

D::D()

B::B()

B::B()

D::D(D& d)

B::B()

A::A(D )

D::~D()

B::~B()

B::B()

D::D()

--2--

D::f()

B::f()

D::g()

--3--

A::g()

B::g()

--4--

B::B()

A::A(A&)

B::B()

A::A()

A::operator= (const A&)

--5--

A::~A()

B::~B()

A::~A()

B::~B()

B::~B()

D::~D()

B::~B()

4.

א.

Script1: how\_many

#!/bin/csh

cat NXN.txt | find\_how\_many

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

script2: find\_how\_many

#!/bin/csh

set count = `wc -l NXN`

set line = ( $< )

while ( $#line != 0)

set words = $#line

if ($words != $count[1] ) then

echo "not NXN"

exit 0

else

set line = ( $< )

endif

end

echo $count[1]

ב.

script1: find\_diag

#!/bin/csh

cat NXN|diag

script2: diag

#!/bin/csh

set wc = `wc -l NXN`

set n = $wc[1]

set count =0

set line = ( $< )

while ( $#line != 0)

@ count++

set word = $line[$count]

echo $word

set line = ( $< )

end