**Курсови задачи по САА**

**Изготвени от Валентин Божидаров Попов, 121221026,**

**ФКСТ, КСИ, II курс, 10 поток, 46А група**

13. Да се проектира структура от данни цикличен списък (динамична реализация), който при нужда удвоява размера си. Да се реализират основните операции за работа с цикличен списък.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

struct node {

int data; //promenliva za stoynostta na elementite v spisuka

int key; //promenliva za index-a na elementite v spisuka

struct node \*next; //pointer, koyto sochi kum sledvashtiq element

};

struct node \*head = NULL; //pointer ot tipa na structurata, koyto sochi kum purviqt element (glavata) na spisuka

struct node \*current = NULL;//pointer ot tipa na structurata, koyto sochi tekushtiqt element (za iterirane)

void printList() { //funkciq za pokazvane na elementite na spisuka

struct node \*ptr = head; //pointer, koyto sochi kum tekushtiq element za obhozhdane

printf("\n[ ");

//start from the beginning

while(ptr != NULL) { //da se vurti dokato elementa, kum koyto sochi pointera e razlichen ot NULL

printf("(%d,%d) ",ptr->key,ptr->data); //nasochvame pointera kum indeksa i kum stoynostta i go printirame

ptr = ptr->next; //nasochvame pointera kum sledvashtiqt element

}

printf(" ]");

}

void insertFirst(int key, int data) { //funkciq za dobavqne na element nay otpred v spisuka

struct node \*link = (struct node\*) malloc(sizeof(struct node)); //zadelqne na neobhodimata pamet za elementa

link->key = key; //nasochvane na pointera kum indeksa, zadavane na stoynost na indeksa po podadeniq argument vuv funkciqta

link->data = data; //nasochvane na pointera kum stoynostta, zadavane na stoynost na stoynostta po podadeniq argument vuv funkciqta

link->next = head; //nasochvane na pointera kum predishniq purvi element

head = link; //zadavame adresa na glavata s noviq purvi element

}

struct node\* deleteFirst() { //funkciq za iztrivane na purviq element vuv spisuka (pop-vane pri stekovete i opashkite)

struct node \*tempLink = head; //zapazvame referenciq kum purviqt element (glavata)

head = head->next; //promenqme pointera na glavata kum sledvashtiqt element, i po tozi nachin predishniq purvi izpada ot psisuka

return tempLink; //vrushtame osvobodeniqt element

}

bool isEmpty() { //proverqva dali spisuka e prazen

return head == NULL; //vrushta true ako pointera na glavata sochi kum NULL, toest spisuka e prazen

}

int length() { //funkciq za dulzhinata na spisuka

int length = 0;

struct node \*current; //tekushtiq element, izpolzvan za obhozhdane na spisuka

for(current = head; current != NULL; current = current->next) { //nasochvame tekushtiq element da zapoche ot glavata, dokato sochi kum element, razlichen ot NULL, da se nasochva kum sledvashtiqt element

length++; //i da se uvelichava dulzhinata na elementa

}

return length; //vrushtame goleminata na spisuka

}

struct node\* find(int key) { //funkciq za namirane na element po index

struct node\* current = head; //nasochvame "iteratora" kum purviqt element na spisuka (glavata)

if(head == NULL) { //pravim proverka dali spisuka ne e prazen

return NULL; //ako e prazen prekratqvame izpulnenieto na funkciqta i vrushtame NULL

}

//navigate through list

while(current->key != key) { //vurtim cikula dokato indeksa na tekushtiqt element ne suvpadne s podadeniqt index na funkciqta

if(current->next == NULL) { //ako dostignem krayniqt element i ne sme namerili turseniqt, prekratqvame izpulnenieto na funkciqta

return NULL; // i vrushtame NULL

} else {

current = current->next; //nasochvame iteratora kum sledvashtiqt element

}

}

return current; //ako sme namerlili elementa s turseniqt index, vrushtame pointera, koyto go sochi

}

struct node\* delete(int key) { //funkciq za iztrivane na element po podaden index

struct node\* current = head; //nasochvane na "iteratora" kum purviqt element (glavata)

struct node\* previous = NULL;//suzdavane na pointer, koyto sochi kum predishniqt element; neobhodimo ni e za da prenasochim pointera kum elementa sled tozi, koyto shte iztriem

if(head == NULL) { //proverka dali spisuka ne e prazen

return NULL; //prekratqvane na izpulnenieto na funkciqta

}

while(current->key != key) { //obhozhdane dokato ne se nameri elementa s podadeniq index

if(current->next == NULL) { //ako sme dostignali do posledniqt element i ne sme namerili suvpadeniq prekratqvame izpulnenieto na funkciqta

return NULL;

} else {

previous = current; //zadavame stoynostta na predishniq element sus adresa na tekushtiqt

current = current->next; //nasochvame "iteratora" kum sledvashtiqt lement

}

}

if(current == head) { //proverka dali elementa koyto iskame da iztriem ne e purviqt (glavata)

head = head->next; //ako iztrivame purviqt element nasochvame pointera na glavata kum sledvashtiqt lement, za da ne izgubim vruzkata

} else {

previous->next = current->next; //izolirame iztritiqt element, kato nasochvame predishniq element, kum sledvashtiqt

}

return current; //vrushtame pointera, koyto sochi kum iztritiqt element

}

void sort() { //funkciq za sortirane na spisuka (izpolzvame buble sort)

int i, j, k, tempKey, tempData;

struct node \*current; //"iterator" za tekushtiq element

struct node \*next; //pointer za sledvashtiqt element

int size = length(); //izvikvame funkciqta za goleminata na spisuka

k = size ;

for ( i = 0 ; i < size - 1 ; i++, k-- ) {

current = head; //nasochvame tekushtiqt element kum glavata

next = head->next; //nasochvame pointera next kum sledvashtiqt element sled glavata

for ( j = 1 ; j < k ; j++ ) {

if ( current->data > next->data ) { //ako tekushtiqt element e po golqm ot sledvashtiqt im razmenqme stoynostite i indeksite posredstvom treta promenliva

tempData = current->data;

current->data = next->data;

next->data = tempData;

tempKey = current->key;

current->key = next->key;

next->key = tempKey;

}

current = current->next; //nasochvame iteratora kum sledvashtiqt element

next = next->next; //nasochvame sledvashtiqt element kum po-sledvashtiqt

}

}

}

void main() {

char command[10];

int n=0, element=0, itemToFound=0, itemToDelete=0; //n - broqt na elementite, koito iskame da vuvedem v spisuka, element - stoynostta na dadeniqt element

do{

printf("command (Add, Print, Delete, Find(by index), DelByIndex, Sort, END): ");

scanf("%s", command);

if(strcmp(command, "Add")==0){

printf("n=");

scanf("%d", &n);

for(int i=1; i<=n; i++){

printf("[%d]:", i);

scanf("%d", &element); //vuvezhdame stoynost na lokalnata promenliva

insertFirst(i,element); //izvikvame funkciqta insertFirst i predavame stoynostta ot lokalnata promenliva element

}

}

else if(strcmp(command, "Print")==0){

printf("Original List: ");

printList(); //izvikvame funkciqta za pechatane na spisuka

printf("\n");

}

else if(strcmp(command, "Delete")==0){

while(!isEmpty()) { //izvikvame funkciqta "isEmpty" za da proverqvame koga spisuka shte se izprazni

struct node \*temp = deleteFirst(); //sled tova izvikvame funkciqta deleteFirst, dokato spisuka e pulen i iztrivame elementite postepenno

printf("Deleted value: (%d,%d) \n",temp->key,temp->data); //pechatame indeksa i stoynostta na iztritiqt element

}

}

else if(strcmp(command, "Find")==0)

{

printf("Item to find: ");

scanf("%d", &itemToFound);

struct node \*foundLink = find(itemToFound); //izvikvame funkciqta za namirane na element po index i prisvoqvame pointera na lokalna promenliva

if(foundLink != NULL) { //ako elementa e nameren

printf("Element found: (%d,%d) ",foundLink->key,foundLink->data); //printirame indeksa i stoynostta na elementa

printf("\n");

} else {

printf("Element not found.\n"); //ako ne e nameren printirame suotvetnoto suobshtenie

}

}

else if (strcmp(command, "DelByIndex")==0){

printf("Item to delete: ");

scanf("%d", &itemToDelete);

delete(itemToDelete); //izvikvame funkciqta za iztrivane na element

printf("List after deleting an item: ");

printList(); //izvikvame funkciqta za printirane na spisuka

printf("\n");

struct node \*foundLink = find(itemToDelete); //izvikvame funkciqta za namirane na elementa i prisvoqvame pointera na lokalna promenliva

if(foundLink != NULL) { //proverqvame, ako elementa ne sochi NULL, e nameren

printf("Element found: (%d,%d) ",foundLink->key,foundLink->data); //printirame indexa i stoynostta na elementa

printf("\n");

} else {

printf("Element not found.\n"); //ako ne go namerim printirame suotvetnoto suobshtenie

}

}

else if(strcmp(command, "Sort")==0){

sort(); //izvikvame funkciqta za sortirane

printf("List after sorting the data: ");

printList(); //i printirame noviq spisuk

printf("\n");

}

}while(strcmp(command, "END")!=0);

}

22. Даден е двумерен масив с n реда и n стълба. Да се намери броя на диагоналите с нулеви елементи, успоредни на вторичния диагонал.

#include <stdio.h>

#include<stdlib.h>

int main(){

int n, br, dolni, gorni, count=0;

printf("n=");

scanf("%d", &n);

int arr[n][n];

for(int i=0; i<n; i++){

for(int j=0; j<n; j++){

printf("[%d][%d]:", i, j);

scanf("%d", &arr[i][j]);

}

}

for(int i=0; i<n; i++){

for(int j=0; j<n; j++){

printf("%d ", arr[i][j]);

}

printf("\n");

}

printf("\n");

br=n+(n-2);

for (int k=0; k<br/2; k++){

dolni=0;

gorni=0;

for(int i=0, j=k; i<=k, j>=0; i++, j--){

if(arr[i][j]!=0){

gorni=1;

}

if(arr[(br/2)-i][(br/2)-j]!=0){

dolni=1;

}

}

if(dolni==0){

count++;

}

if(gorni==0){

count++;

}

printf("\n");

}

printf("broq na diagonalite s nulevi elementi, usporedni na vtorichniq e: %d", count);

return 0;

}

n - броя на редовете и колоните

br - броя на диагоналите, успоредни на вторичния(без него)

dolni - флаг за ненулеви елементи за диагонали под вторичния

gorni - флаг за ненулеви елементи за диагонали над вторичния

count - броя диагонали с нулеви елементи, успоредни на вторичния

намиране на броя br:

-> при n=2 => 2 диагонала

-> при n=3 => 4 диагонала

-> при n=4 => 6 диагонала

-> при n=5 => 8 диагонала...

=> br=n+(n-2)

пример:

3 2 0 5 2

6 0 6 3 0

0 7 4 0 2

8 5 0 6 5

6 0 1 2 0

диагонали успоредни на вторичния:

1. 3 => [0][0]

2. 0 => [(br/2)-i][(br/2)-j] => [4][4]

3. 2, 6 => [0][1], [1][0]

4. 2, 5 => [4][3], [3][4]

5. 0, 0, 0 => [0][2], [1][1], [2][0]

6. 1, 6, 2 => [4][2], [3][3], [2][4]

7. 5, 6, 7, 8 => [0][3], [1][2], [2][1], [3][0]

8. 0, 0, 0, 0 => [4][1], [3][2], [2][3], [1][4]

n=5; br=8

k=0; k<br/2:

i=0, j=k; i<=k, j>=0; i++, j--:

k=0: (3 & 0)

i=0; j=0: [0][0] - 3, [(br/2)-i][(br/2)-j] => [4][4] - 0

k=1: (2, 6 & 2, 5)

i=0; j=1: [0][1] - 2, [4][3] - 2

i=1; j=0: [1][0] - 6, [3][4] - 5

k=2: (0, 0, 0 & 1, 6, 2)

i=0; j=2: [0][2] - 0, [4][2] - 1

i=1; j=1: [1][1] - 0, [3][3] - 6

i=2; j=0: [2][0] - 0, [2][4] - 2

k=3: (5, 6, 7, 8 & 0, 0, 0, 0)

i=0; j=3: [0][3] - 5, [4][1] - 0

i=1; j=2: [1][2] - 6, [3][2] - 0

i=2; j=1: [2][1] - 7, [2][3] - 0

i=3; j=0: [3][0] - 8, [1][4] - 0

след това ако в даден диагонал срещне ненулев елемент, флага става равен на 1, и след като приключи проверката на текущия диагонал, count не се увеличава!

Ако пък при проверка на диагонала не е срещнат ненулев елемент, флага си остава равен на 0, и след края на обхождането на текущия диагонал count се увеличава с 1!

->пример 1:

вход:

1 2 3

4 5 6

7 8 9

изход:

broq na diagonalite s nulevi elementi, usporedni na vtorichniq e: 0

->пример 2:

вход:

3 2 0 5 2

6 0 6 3 0

0 7 4 0 2

8 5 0 6 5

6 0 1 2 0

изход:

broq na diagonalite s nulevi elementi, usporedni na vtorichniq e: 3