САА - Упражнение 5 (Лаб\_5\_САА)

Задача 49

Стек

#include "stdafx.h"

#include <iostream>

using namespace std;

typedef int data;

#define MAX 10

data stack[MAX];

int top;

void init()

{

top = 0;

}

void push(data i)

{

if(MAX == top)

cout << "Prepalvane na steka! \n";

else

stack[top++] = i;

}

data pop()

{

if(0 == top)

{

cout << "Stekat e prazen! \n";

return 0;

}

else

return stack[--top];

}

int isEmpty()

{

return(0 == top);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

data p;

init();

cin >> p;

while(0 != p)

{

push(p);

cin >> p;

}

while(!isEmpty())

cout << " " << pop();

cout << "\n";

return 0;

}

Задача 50

Опашка

#include "stdafx.h"

#include <iostream>

using namespace std;

typedef int data;

#define MAX 10

data queue[MAX];

int front, rear, empty;

void init()

{

front = rear = 0;

empty = 1;

}

void put(data i)

{

if(front == rear && !empty)

{

cout << "Prepalvane! \n";

return;

}

queue[rear++] = i;

if(rear >= MAX)

rear = 0;

empty = 0;

}

data get()

{

data x;

if(empty)

{

cout << "Opashkata e prazna! \n";

return 0;

}

x = queue[front++];

if(front >= MAX)

front = 0;

if(front == rear)

empty = 1;

return x;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

data p;

int i;

init();

for(i = 0; i < 2\*MAX; i++)

{

put(i);

p = get();

cout << " " << p;

}

cout << "\n";

for(i = 0; i < MAX+1; i++)

put(i);

for(i = 0; i < MAX+1; i++)

get();

return 0;

}

XIII.Двоични дървета

Задача 51

Основни операции с двоични дървета

#include "stdafx.h"

#include <iostream>

using namespace std;

typedef char \*data;

typedef int keyType;

struct tree {

keyType key;

data info;

struct tree \*left;

struct tree \*right;

};

/\*Tarsene v binarno darvo\*/

struct tree \*search(keyType key, struct tree \*T)

{

if(NULL ==T )

return NULL;

else if(key < T->key)

return search(key, T->left);

else if(key > T->key)

return search(key, T->right);

else

return T;

}

/\*Vkluchvane v binarno darvo\*/

void insertKey(keyType key, data x, struct tree \*\*T)

{

if(NULL == \*T)

{

\*T = (struct tree \*)malloc(sizeof(\*\*T));

(\*T)->key = key;

(\*T)->info = x;

(\*T)->left = NULL;

(\*T)->right = NULL;

}

else if(key < (\*T)->key)

insertKey(key, x, &(\*T)->left);

else if(key > (\*T)->key)

insertKey(key, x, &(\*T)->right);

else

cout << "Elementat e veche v darvoto!\n";

}

/\*Namirane na min element v darvo\*/

struct tree \*findMin(struct tree \*T)

{

while(NULL != T->left)

T = T->left;

return T;

}

/\*Izkluchvane ot binarno darvo\*/

void deleteKey(keyType key, struct tree \*\*T)

{

if(NULL == \*T)

cout << "Varhat, koito triabva da se izkluchi lipsva!\n";

else

{

if(key < (\*T)->key)

deleteKey(key, &(\*T)->left);

else if(key > (\*T)->key)

deleteKey(key, &(\*T)->right);

else

if((\*T)->left && (\*T)->right)

{

struct tree \*replace = findMin((\*T)->right);

(\*T)->key = replace->key;

(\*T)->info = replace->info;

deleteKey((\*T)->key, &(\*T)->right);

}

else

{

struct tree \*temp = \*T;

if((\*T)->left)

\*T = (\*T)->left;

else

\*T = (\*T)->right;

free(temp);

}

}

}

void printTree(struct tree \*T)

{

if(NULL == T)

return;

cout << T->key << " ";

printTree(T->left);

printTree(T->right);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

struct tree \*T = NULL, \*result;

int i;

/\*Vkluchva 10 varha s proizvolni kluchove\*/

for(i=0; i<10; i++)

{

int ikey = (rand()%20) + i;

cout << "Vmakva se element s kluch " << ikey << "\n";

insertKey(ikey, "someinfo", &T);

}

cout << "Darvo: ";

printTree(T);

cout << "\n";

/\*pretarsva za element s kluch 9\*/

result = search(9, T);

cout << "Nameren e: " << result->info << "\n";

/\*iztriva proizvolni 10 varha ot darvoto\*/

for(i=0; i<10; i++)

{

int ikey = (rand()%20) + i;

cout << "Iztriva se element s kluch " << ikey << "\n";

deleteKey(ikey, &T);

}

cout << "Darvo: "; printTree(T); cout << "\n";

return 0;

}

Задача 52

Програма за построяване на оптимално двоично дърво по алгоритъма на Хъфман и получаване на кодовете на буквите

#include "stdafx.h"

#include <iostream>

#include "string.h"

using namespace std;

typedef unsigned char byte;

char text[] = "Sintez i Analiz na Algoritmi";

int text\_len;

struct Huf {

byte id;

int wh;

Huf \*left, \*right;

};

struct List {

List \*next;

Huf \*tree;

};

List \*head;

char code[256];

void createList();

void writeList();

void delList(List \*);

void addList(Huf \*);

Huf \*findDels();

void createTree();

void rlrootTree(Huf \*, unsigned);

int \_tmain(int argc, \_TCHAR\* argv[])

{

text\_len = strlen(text);

createList();

writeList();

createTree();

cout << "writeCodes\n";

rlrootTree(head->tree, 0);

cout << endl;

return 0;

}

void createList()

{

int i;

int ch[256] = {0};

for (i=0; i<text\_len; i++)

ch[text[i]]++;

List \*l;

Huf \*h;

head = 0;

for (i=0; i<255; i++)

if (ch[i]>0)

{

h = new Huf;

h->id = i;

h->wh = ch[i];

h->left = 0;

h->right = 0;

l = new List;

l->tree = h;

l->next = head;

head = l;

}

}

void writeList()

{

cout << "writeList\n";

List \*l = head;

while(l)

{

cout << (l->tree)->id << " ";

l = l->next;

}

cout << endl;

l = head;

while(l)

{

cout << (l->tree)->wh << " ";

l = l->next;

}

cout << endl;

}

void delList(List \*l)

{

List \*lp, \*lc;

if (l==head)

{

head=l->next;

delete l;

}

else

{

lp = head;

lc = lp->next;

while(lc!=l)

{

lp = lc;

lc = lc->next;

}

lp->next = lc->next;

delete lc;

}

}

void addList(Huf \*h)

{

List \*l = new List;

l->tree = h;

l->next = head;

head = l;

}

Huf \*findDels()

{

List \*l = head, \*sm = head;

Huf \*h;

while(l)

{

if ((l->tree)->wh < (sm->tree)->wh)

sm = l;

l = l->next;

}

h = sm->tree;

delList(sm);

return h;

}

void createTree()

{

Huf \*h, \*h1, \*h2;

while(head->next)

{

h1 = findDels();

h2 = findDels();

h = new Huf;

h->id = ' ';

h->wh = h1->wh + h2->wh;

h->left = h1;

h->right = h2;

addList(h);

}

}

void rlrootTree(Huf \*h, unsigned index)

{

if(h)

{

code[index] = '0';

rlrootTree(h->right, index+1);

if(h->left == 0)

{

code[index] = '\0';

cout << h->id << "->" << code << " ";

}

code[index] = '1';

rlrootTree(h->left, index+1);

}

}

XIV.Вероятностни алгоритми

Основно свойство на алгоритмите е детерминираността, което означава, че за всеки блок от даден алгоритъм, при едни и същи входни данни всеки път се получават едни и същи изходни данни. Ако поне на едно място в алгоритъма има случаен избор, тогава алгоритъмът не е детерминиран и се нарича вероятностен.

Числени вероятностни алгоритми

Генерира се едно случайно решение, а при всяка следваща итерация се достига до все по-точно решение.

Вероятностни алгоритми Монте Карло

Винаги дават отговор, но резултата не винаги е верен. Ако отговорът е положителен, то и резултата ще бъде верен. Отрицателният отговор обаче не гарантира липса на решение. Предимството на този вид вероятностни алгоритми е тяхното бързодействие.

Вероятностни алгоритми Лас Вегас

Ако дадат отговор, то той винаги е верен, но има случаи, когато не могат да намерят решение.

Вероятностни алгоритми Шерууд

Детерминираните алгоритми винаги дават отговори и решението винаги е вярно, но е възможно времето за "най-тежкия" случай да е много отдалечено от Тср и Тmin. При алгоритмите Шерууд тази разлика се намалява.

Примери на вероятностни алгоритми

Пример 1

Разглеждаме мишена, в която стреляме и броим колко пъти попадаме в кръга и колко пъти извън него. Броят на попаденията е пропорционален на площта и следователно ако к са попаденията в кръга, а n са общия брой изстрели, то к/n ≈ πr2/4r2 = π/4, т.е. изчисляваме π/4.

Пример 2

Задачата за осемте царици. Да се разположат осем царици по шахматната дъска, така че да не се "бият" една с друга.

Решение с връщане. Когато сме достигнали определена позиция, знаем кои полета не са били "бити", можем да отидем на определен ред и ако достигнем ситуация, при която всички са били "бити" и няма къде да сложим следващата царица, се връщаме и слагаме фигурата на друго поле.

Решение с генератор на случайни числа. Знаейки кои полета не са били "бити", с генератор на случайни числа избираме едно от тях и анализираме докато намерим или не намерим решение. След това процедурата се повтаря отначало.

Пример 3

Задача за изборът на 13 произволни карти при игра на бридж. Тази задача спада към задачите за избор на М елемента от N съществуващи.

Един подход за решаването на тази задача е обхождането на всички N елемента и избор на всеки от М-те елемента с вероятност M/N. Този подход е сравнително бавен и не гарантира, че след обхождането на N-те елемента, ше бъдат избрани точно M на брой.

Друг подход е генерирането на M на брой случайни числа и избирането само на тези от тях, които до момента не са били избрани, т.е., в множеството от M числа да няма повтарящи се. Този подход също е бавен, например, ако М = 98, а N = 100 и вече са били избрани 97 числа.

Генератори на случайни числа

Вероятностните алгоритми използват програмни генератори на псевдослучайни числа. Генераторите на случайни числа са програми, които образуват поредица от случайни числа, в зададен интервал, при определен закон на разпределение на плътността на вероятностите. В истинският смисъл на думата генератор на случайни числа практически не се реализира - в даден момент от време стойностите, които се получават са равномерно разпределени. Говорим за псевдослучайни генератори на случайни числа.

Най-често се реализират и използват програми-генератори на равномерно разпределени случайни числа в интервала [0...1), или в зададен числен интервал [0, m-1]. В случая, често се използва формулата на Кнут:

Yi=(A\*Yi-1+C) mod M,

където А и С са подходящо избрани константи,Yo е начално число, което не се използва, но служи за “стартиране” на генерирането на следващите, вече случайни числа.

<https://en.wikipedia.org/wiki/Randomized_algorithm>

Задача 53

Генериране на N на брой цели случайни числа

#include "stdafx.h"

#include <iostream>

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[])

{

int i,a, c, n, m, y0, y;

cout << "\nVavedete broya n na generiranite sluchayni chisla: ";

cin >> n;

cout << "\nVavedete gornata granica na intervala m: ";

cin >> m;

cout << "\nVavedete stojnost za a: ";

cin >> a;

cout << "\nVavedete stoynost za c: ";

cin >> c;

cout << "\nVavedete nachalna stoynost yo: ";

cin >> y0;

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

{

y = (a\*y0+c) % m;

cout << "\n";

cout << i << "-toto sl. chislo e : " << y;

}

cout << "\n";

return 0;

}

XV.Генетични алгоритми

В основата на тези алгоритми стои стествения подбор. В началото се генерират случайни решения – родители. От тях се генерират нови решения, като новите се оценяват по определени критерии и се вземат по-добрите. Продължава се с определен брой цикли, докато се намери приблизително решение.

1) Генериране на n случайни пермутации. (примерно за задачата за търговския пътник)

От две вече генерирани пермутации се избира произволна точка и двете части се разменят.

2) Селекция – избират се пермутациите с най-малък път .

3) Условие за край – проверени са 10000 ÷ 40000 пермутации.



<http://www.obitko.com/tutorials/genetic-algorithms/bulgarian>

Задача 54

Да се състави програма, чрез която от клавиатурата се въвежда естествено число N от интервала [1...100]. Програмата да генерира N броя естествени числа от интервала [1...999]. Числата да бъдат записани в списък/стек. При извеждането на елементите от стека числата да се разделят на четни и нечетни, като първо се извеждат всички четни, а след това всички нечетни числа. Четните стойности се извеждат в същия ред, по който са били въведени, а нечетните стойности се извеждат в ред обратен на въвеждането им.

#include "stdafx.h"

#include <iostream>

#include <list>

#include <stack>

#include <ctime>

using namespace std;

list<int> m\_list;

stack<int> m\_stack;

void pylni\_spis(int N)

{

int i, chis;

cout << "Generiram " << N << " broq psewdosluchajni chisla:\n";

for (i=1; i<=N; i++)

{

chis = 1+rand()%1000;

cout << chis << " ";

if (chis%2)

m\_list.push\_back(chis); // zapis na nechetna stojnost w spisyk

else

m\_stack.push(chis);//zapis na chetna stojnost w steka

}

cout << endl;

//cout izwevda elementite ot steka i gi zapiswa w spisyka

while (!m\_stack.empty())

{

chis = m\_stack.top();

m\_list.push\_back(chis);

m\_stack.pop();//maha element ot wyrha

}//izwevda wsichki chetni stojnosti ot steka i gi wywevda w spisyka

}//pylni\_spis(int N)

void pechat\_spis()

{

list<int>::reverse\_iterator nom;

int chis;

while (!m\_list.empty())

{

chis = m\_list.back();//tekushiqt posleden element

cout << " " << chis;

m\_list.pop\_back();//maha element ot kraq na spisyka

}

cout << endl;

}//pechat\_spis

int \_tmain(int argc, \_TCHAR\* argv[])

{

time\_t t;//polzwa tekushoto wreme

srand(time(&t));//inicializira generator na suchajni chisla

int N;

cout << "Wywedete broj na chislata: ";

cin >> N;

pylni\_spis(N); // wywevda psewdosluchajni stojnosti

cout << "Spisykyt sydyrva slednite "<<N<<" chisla:\n";

pechat\_spis();//izwevda elementite na spisyka

//system("pause");

return 0;

}

Задача 55\*

Съставете алгоритъм и напишете програма за търсене на елемент в предварително подреден едномерен масив с неповтарящи се елементи. По зададена стойност, програмата да извежда индекса на елемента. Да се използва метода Разделай и Владей.

Задача 56\*

Съставете алгоритъм и напишете програма за определяне честотата на срещане на елементите на двумерен масив. Елементите на масива да бъдат естествени едноцифрени числа.

Задачи за изпълнение

1. Изпълнете, разгледайте и анализирайте решените задачи – номера 49, 50, 51, 52, 53 и 54.

2. Съставете алгоритъм и напишете програмен код на нерешените задачи – номера 55\* и 56\*.