Assignment

Filip Alina-Andreea, CR 1.2.A

May 19, 2019

1 Cerinta

N copii joaca bine-cunoscutul joc ala-bala-portocala. Ei sunt asezati intr-un cerc si incep sa numere de la 1 la k. Al k-lea copil este scos din joc. Procesul continua pana cand toti copii sun eliminati din joc. Implementeaza un algoritm eficient care sa printeze ordinea in care copii sun eliminati din joc folosind o lista circulara.

Problema ne cere sa afisam ordinea in care copii sunt scosi din joc.

2 Algoritmi

Pentru realizarea acestui program am utilizat mai multi algoritmi impartiti in mai multe coduri sursa.

2.1 Algoritmul de adaugare a unui element la finalul unei liste

PUSH-ELEMENT-END(List,n)

1. i=head;
2. while (i.next != head)
3. i = i.next;
4. last= i;
5. last.next = new;
6. new.info = n;
7. new.next = head;

Algoritmul adauga la finalul listei cate un nou element de fiecare data cand este apelat. Urmeaza urmatorii pasi:

- Face iteratorul sa fie egal cu head-ul care pointeaza catre primul element din list.

- Cat timp elementul urmator este diferit de capul liste executa urmatoarele:
- Iteratorul primeste valoarea urmatorului element pana la intalnirea capului listei.
- Ultimul element primeste valoarea iteratorului.
- Se creeaza o legatura intre ultimul element si noul nod.
- Noul nod primeste valoarea pe care dorim sa o adaugam.
- Se creeaza legatura dintre noul nod care acum a devenit ultimul nod din lista si capul listei.

Acesta este un algoritm implementat dupa modelul celui predat la curs.

Algoritmul are o complexitate egala cu :

$$T(n) = 1 + n(1+1) + 1 + 1 + 1 + 1 = O(n)$$

2.2 Algoritmul de extragere a unui element din lista

POP-ELEMENT-AT-POSITION(List, p)

1. assert(p mai mare = 0);
2. i = head;
3. cp=0;
4. while(cp ¡ p)
5. cp++;
6. i = i.next;
7. if(i==head)
8. a++;
9. while((a!=0)and(a!=1))
10. if(i.next==head)
11. i=i.next.next
12. a–;
13. else
14. i=i.next;
15. a–;
16. if(i.next==head)

17. i=i.next;
18. prev = i;
19. poped = prev.next;
20. prev.next = poped.next;
21. aux = poped.info;
22. return aux;

Algoritmul extrage elementul de pe pozitia ceruta atunci cand se ajunge la acea pozitie. El functioneaza astfel:

- Assert pus la inceputul unui program ii specifica compilatorului conditiile sub care un programator se asteapta ca functioneze programul. In avest caz ne asteptam ca variabila p sa fie mereu mai mare sau cel putin egala cu 0.
- Iteratorul ia valoarea capului listei.
- Pozitia curenta ia valoarea 0, acolo unde se afla capul.
- Cat timp pozitia curenta este mai mica decat pozitia ceruta (cp¡p) executa
- Pozitia curenta avanseaza cu o unitate.
- I ia valoarea urmatorului element din lista.
- Daca i este egal cu head acreste.Am intodus acest if pentru a monitoriza de cate ori apare head-ul pana la ajunge in pozitia dinaintea pozitiei cerute.
- Cat timp a este diferit de 1 si 0 (am adaugat si 1 deoarece am observat ca daca head-ul apare de n ori atunci decalarea este defapt de n-1 pozitii) atunci executa:
- Daca urmatorul element este head atunci de avanseaza cu 2 pozitii pentru a sari de direct la primul element din lista, urmata de scaderea lui a.
- In caz contrat se avanseaza doar o data in lista, apoi se scade a.
- Prev , elementul care precedeaza elementul de pe pozitia pe care vrem sa o eliminam ia valoarea lui i.
- Elementul pe care vrem sa il eliminam ia valoarea urmatorului element dupa prev.
- Se creaza o legatura intre prev si elementul care urmeaza dupa elementul pe care vrem sa il scoatem din lista, adica prev nu mai pointeaza spre poped si pointeaza spre urmatorul element dupa poped.
- Aux ia valoarea ce se regaseste pe pozitia eliminata.

- Algoritmul returneaza valoarea lui aux;

Complexitatea algoritmului este:

$$T(n) = 3 + n(1 + 1 + 1 + m + 1) + (a - 1)(3m + 3p + 1) + 6 = O(n)$$

a-1mereu mai mic decat n; m, p mereu mai mic decat a-1; m reprezinta numarul de dati cand se intra pe prima ramura a lui if si p pe a 2-a ramura In cel mai bun caz, complexitatea algoritmului poate fi O(1) cand lista si se cere elementul de pe pozitia 0. In cel mai rau caz, complexitatea algoritmului poate fi $O(n^2)$ atunci cand lisat nu are nici un element si se cere un element de pe o pozitie diferita de 0.

2.3 Algoritmul de printare a listei

PRINT-LIST(list)

1. i = head
2. while (i.next != head)
3. printeaza i.next.info
4. i=i.next

Algoritmul realizeaza afisarea listei element cu element pana la intalnirea lui capului de lista. Algoritmul urmeaza urmatorii pasi:

- I primeste valoarea capului de lista.
- Cat timp elementul urmator dupa i este diferit de capul listei executa:
- Afisarea valorii aflate in nodul urmator.
- I primeste valoarea urmatorului nod sau trece pe pozitia urmatorului nod. Nodul afisat este mereu urmatorul dupa nodul curent.

Vectorul este modificat la fiecare incrementare a lui i si fiecare interschimbare intre elemente iar la final este transmis rezultatul final.

Complexitatea algoritmului este :

$$T(n) = 1 + n(2 + 1) = O(n)$$

In cel mai bun caz, complexitatea algoritmului este O(1) cand lista nu are nici un element (vida).

2.4 Algoritmul principal

MAIN()

1. for(j=1;j¡=10;j++)
2. if(k==1)
3. for(i=1;i¡=nr-copii;i++)
4. aux=pop-element-at-position(head,0);
5. if(k¡=nr-copii)
6. i=k-1;
7. while (i!=nr-copii)
8. aux = pop-element-at-position(head,k-1);
9. i++;
10. nr-copii=k-1;
11. for(i=1; i¡=nr-copii; i++)
12. aux = pop-element-at-position(head,k);
13. return 0;

Am ales sa scriu doar codul care este mai important din tot cel existent in program si codul care are nevoie de explicatii.

m=10 reprezinta numarul testului care se ia in considerare si al fisierului in care se scrie (test1.in si test1.out).

Daca k sete egal cu 1 inseamna ca din lista se scoate mereu rpimul element.

Daca k este mai mic decat numarul de copii atunci in seamna ca de la pozitia k inclusiv se scot pe rand elementele pana la finalul listei. In functie este specificat k-1 deoarece in lista elementele in cep cu numerotarea de la 0 adica daca in lista avem 5 elemente atunci ultima pozitie este 4 iar prima este 0. Dupa terminarea listei mai raman elementele pana la k care reprezinta numarul de copii ramasi.

Pentru copii ramasi sau daca k este mai mare decat numarul de copii se parcurge lista de mai multe ori pana se ajunge la k si se elimina elementul respectiv.

3 Date experimentale

Datele experimentale au fost create cu uneui program ce scrie date random in 10 fisiere sursa ce sunt folosite la inceputul programului ca date de intrare.

3.1 Algoritmul de randomizare a datelor de intrare

Acest algoritm se gaseste in main

1. for(j=1;j¡=10;j++)
2. sprintf(temp-name-in, "teste
   test-procentd.in", j+1);
3. FILE* test-in = fopen(temp-name-in, "wt");
4. nr-copii=rand();
5. k=rand();
6. if(nr-copii mai mic = 0 —— nr-copii ==1)
7. nr-copii=rand();
8. if(kmai mic=0)
9. k=rand()
10. fprintf(test-in, " procentd", nr-copii);
11. fprintf(testi-n, "procentd", k);
12. fclose(test-in);

Daca numarul de copii este egal cu 0 sau 1 atunci jocul nu se poate juca deci se alege alta valoare. Daca k este mai mic decat 0 sau egal cu 0 atunci nu se pot lua cate k copii din lista. Fisierele se regasesc impreuna cu codul sursa intr-un fisier numit teste. Nr-copii primeste de 10 ori valori random pana la 100, fiecare valoare fiind scrisa in fisierul sursa. Se creaza 10 fisiere sursa. Teste-in are rol de identificator al fisierului in program. Fopen deschide fisierul iar wt inseamna ca in program se poate doar scrie in fisier iar daca fisierul nu este creat atunci se creaza in momentul deschiderii.

4 Rezultate si concluzii

Acest program este unul foarte interesant pentru mine deoarece am reusit sa vad cum functioneaza o lista circulara si am invatat sa folosesc fisierele si sa le creez, sa scriu in ele si sa citesc si sa folosesc datele din interiorul unui fisier.

Am avut probleme pana sa imi dau seama ca numerotarea pozitiilor in lista se decaleaza atunci cand se intalneste head si a fost destul de greu sa modific programul pentru a aduce totul asa cum ar trebui sa fie si asa cum imi iesea mie pe foaie.

Multe functii folosite au fost preluate din cele pe care le-am studiat la laborator, eu doar le-am facut niste modificari unde era necesar.

Am adaugat niste cazuri pentru input-ul nepotrivit pentru a salva din timpul de executie.

In concluzie, problema a fost interesanta deoarece face ceva ce in realitate pare foarte usor insa atunci cand vrei sa implementezi un program care face acel lucru s-ar putea sa nu fie chiar asa de usor cum am credea.