Tema 3 - Stive, cozi, liste înlănţuite

Probleme de 3p

1. Se dă un şir de paranteze deschise şi închise rotunde. Să se verifice dacă şirul este corect. Pentru rezolvare folosiţi o stivă de catactere implementare proprie sau stack din c++.**Exemplu**: şirul (()()) este corect, şirul ((() nu este corect, şirul ())(nu este corect.

2. Să se simuleze o stivă utilizând două cozi. Se va crea structura stivă cu funcţiile de push(), pop(), isEmpty() şi clear(), iar cele 2 cozi vor fi câmpuri ale structurii. Puteţi folosi implementare proprie de coadă sau queue din C++.

3. Să se implementeze o listă dublu înlănţuită cu funcţionalităţile descrise în continuare. Se cere utilizarea unei structuri *node* care are trei câmpuri: un câmp pentru informaţie (de tip int) şi două câmpuri de tip pointer la *node* pentru legăturile către elementele precedent şi următor. Se cere utilizarea unei structuri *List* care are ca membri două variabile de tip $node*$ reprezentând primul respectiv ultimul element din listă, o variabilă de tip int reprezentând numărul de elemente din listă şi funcţiile:

   - $push\_front(key)$ - adaugă cheia *key* în capul listei
   - $push\_back(key)$ - adaugă cheia *key* la finalul listei
   - $pop\_front()$ - şterge primul element din listă
   - $pop\_back()$ - şterge ultimul element din listă
   - $find(key)$ - caută o cheie *key* în listă - returnează pointer la nodul cu cheia *key* sau NULL
   - $clear()$ - şterge toate elementele listei
   - $print()$ - afişează elementele listei

   În funcţia *main* creaţi un scurt context de testare şi apelaţi funcţiile implementate.

4. Să se implementeze o listă dublu înlănţuită cu funcţionalităţile descrise în continuare. Se cere utilizarea unei structuri *node* care are trei câmpuri: un câmp pentru informaţie (de tip int) şi două câmpuri de tip pointer la *node* pentru legăturile către elementele precedent şi următor. Se cere utilizarea unei structuri *List* care are ca membri două variabile de tip $node*$ reprezentând primul respectiv ultimul element din listă, o variabilă de tip int reprezentând numărul de elemente din listă şi funcţiile:

   - $push\_back(key)$ - adaugă cheia *key* la finalul listei
   - $remove(int\ key)$ - şterge toate apariţiile cheii *key* (implică căutare)
   - $erase(node*Nod)$ -şterge un element Nod din listă (NU implică căutare)
   - $size()$ - returnează numărul de elemente din listă.
   - $print()$ - afişează elementele listei

   În funcţia *main* creaţi un scurt context de testare şi apelaţi funcţiile implementate.

5. Să se implementeze o listă dublu înlănţuită cu funcţionalităţile descrise în continuare. Se cere utilizarea unei structuri *node* care are trei câmpuri: un câmp pentru informaţie (de tip int) şi două câmpuri de tip pointer la *node* pentru legăturile către elementele precedent şi următor. Se cere utilizarea unei structuri *List* care are ca membri două variabile de tip $node*$ reprezentând primul respectiv ultimul element din listă, o variabilă de tip int reprezentând numărul de elemente din listă şi funcţiile:

   - $push\_back(key)$ - adaugă cheia *key* la finalul listei
   - $insert(node*Nod, int\ val)$ - inserează un element cu cheia *val* înainte de nodul indicat de Nod.
   - funcţia $size()$ - returnează numărul de elemente din listă.
   - $merge(ListaL)$ - concatenează lista curentă cu lista L. După apel lista curentă conţine elementele celor două liste concatenate, iar L devine vidă.
   - $print()$ - afişează elementele listei

   De asemenea să se implementeze următoarele funcţii, care nu fac parte din structură:

   - $palindrom(Lista\ L)$ - verifică dacă lista este palindrom (0.5 p)
   - $compare(Lista\ L1,\ ListaL2)$ - returnează 1 dacă $L1$ şi $L2$ sunt identice şi 0 altfel. (0.20 p)

În funcţia *main* creaţi un scurt context de testare şi apelaţi funcţiile implementate.

6. Să se implementeze o listă dublu înlănţuită cu funcţionalităţile descrise în continuare. Se cere utilizarea unei structuri *node* care are trei câmpuri: un câmp pentru informaţie (de tip int) şi două câmpuri de tip pointer la *node* pentru legăturile către elementele precedent şi următor. Se cere utilizarea unei structuri *List* care are ca membri două variabile de tip $node*$ reprezentând primul respectiv ultimul element din listă, o variabilă de tip int reprezentând numărul de elemente din listă şi funcţiile:

   - $push\_back(key)$ - adaugă cheia *key* la finalul listei
   - $palindrom(List\ L)$ - verifică dacă lista este palindrom
   - $compare(List\ L1, ListL2)$ - returnează true dacă $L1$ şi $L2$ sunt identice şi false altfel.
   - $print()$ - afişează elementele listei

   În funcţia *main* creaţi un scurt context de testare şi apelaţi funcţiile implementate.

Probleme de 4p

7. Se dă un şir de paranteze deschise şi închise de tip (,), [,], {, }. Să se verifice dacă şirul este corect. Pentru rezolvare folosiţi o stivă de catactere implementare proprie sau std::stack. **Exemplu**: şirul [()()] este corect, şirul ([]) nu este corect, şirul ()](nu este corect.

8. Să se implementeze o coadă circulară. Se va crea structura coadă cu funcţiile de push(), pop(), isEmpty() şi clear(), această structură va conţine un vector de dimensiune fixă. Algoritmul este prezentat în materialul de curs, precum şi în materialul bibliografic recomandat.

9. Să se creeze o structura *UndoRedo* care va reţine ordinea în care au fost apelate mai multe opţiuni, fiecare opţiune este reprezentată de un număr natural. Structura va avea următoarele funcţii:

   - *add* - înregistrează o nouă opţiune
   - *get()* - returnează indicele ultimei opţiuni
   - *undo()* - elimină ultima opţiune adăugată

- *redo()* - adaugă ultima opţiune ştearsă

Aceste funcţii se pot apela în orice ordine. Pentru implementare este necesară utilizarea unei stive.

10. Un anumit etaj al unei clădiri este reprezentat schematic sub forma unei matrice ce conţine valorile -1 şi 0, unde -1 reprezintă zid şi 0 reprezintă spaţiu liber. Pereţii sunt de grosime 1 şi uşile nu sunt marcate (se consideră tot perete). Determinaţi numărul de încăperi ale etajului respectivşi cu suprafaţa maxima a unei camere utilizând o coadă.
    În verificarea apartenenţei la aceeaşi cameră în matrice se consideră doar vecinătăţile pe orizontală şi verticală.
    Evitaţi utilizarea recursivităţii. Puteti folosi queue din C++, în coadă reţineţi acele poziţii pe care le-aţi vizitat.

    Exemplu:

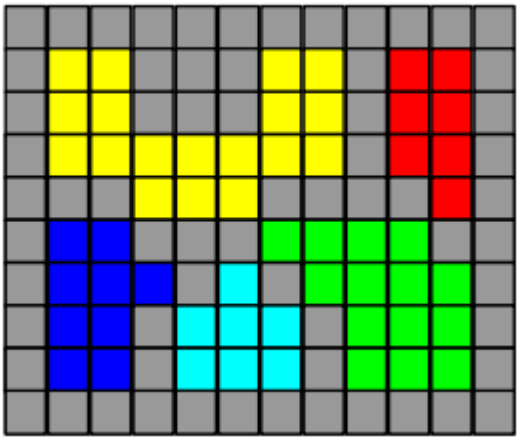

Figure 1: a. Există 5 camere iar suprafaţa celei mai mari este 18.

11. Se consideră o stivă(implementare proprie sau din c++). Să se sorteze astfel încât elementele acesteia să fie crescătoare privind de la bază către vârf. În vederea implementării se pot utliza doar stive auxiliare(una de preferat).

Probleme de 5p

12. Se consideră un labirint reprezentat printr-o matrice în care zidurile sunt marcate prin -1 şi drumurile prin 0. În labirint se află un şoricel pe poziţia (x_0, y_0) şi o bucată de brânză pe poziţia (x_1, y_1). Să se găsească un drum (de preferinţă de lungime minimă) de la şoricel la brânză. Nu utilizaţi recursivitate! Utilizaţi o coadă. Puteti folosi queue din C++.

13. Un anumit etaj al unei clădiri este reprezentat schematic sub forma unei matrice ce conţine valorile -1 şi 0, unde -1 reprezintă zid şi 0 reprezintă spaţiu liber. Pereţii sunt de grosime 1 şi uşile nu sunt marcate (se consideră tot perete). Care perete poate fi dărâmat (o poziţie maractă cu -1 se va marca cu 0) a. î. să se obţină o încăpere de suprafaţă maximă? Evitaţi utilizarea recursivităţii. Puteti folosi queue din C++.

    Exemplu:

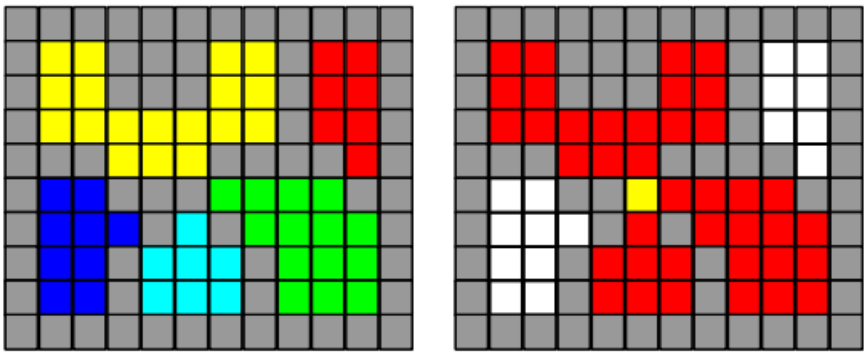

Figure 2: Dacă se şterge elementul de zid marcat cu galben se obţine cea mai mare cameră posibilă.
Se observă că există 5 camere, fiecare marcate cu altă culoare în figura 2a. Camera marcată cu galben are suprafaţa de 18 unităţi, camera roşie de 7 unităţi, camera albastră de 9 unităţi, camera cyan de 7 unităţi şi camera verde de 14 unităţi.
În figura 2b. se observă faptul că, dacă se elimină zidul de pe poziţia marcată cu galben, se unesc trei încăperi şi se obţine ce mai mare încăpere posibilă, cu suprafaţa de 40 de unităţi.