Tema 1 – LAN PARTY

TUDOR COSTIN-CRISTIAN – 314AB

Pentru a rezolva tema, am implementat un set de functii si structuri de date, acestea fiind explicate mai jos:

Structura PLAYER: Defineste un jucator prin nume, prenume(pointeri catre siruri de caractere) si xp(int).

Structura TEAM: Defineste o echipa prin nume(pointer catre un sir de caractere), size(de tip int, numarul de jucatori care fac parte din acea echipa), punctaj(float, punctajul echipei – media aritmetica a experientei jucatorilor), player(pointer catre un tablou unidimensional de tip PLAYER, tabloul contine informatiile despre toti jucatorii din echipa) si next(pointer catre urmatoarea echipa din lista)

Structura MATCH: Defineste un meci dintre doua echipe distincte. Aceasta contine doi pointeri catre valori de tip TEAM(team1, team2) si next(pointer catre urmatorul meci).

Structura QUEUE: Defineste coada in care vor fi puse meciurile prin front(primul meci din coada) si rear(ultimul meci din coada).

Structura NODE: Defineste un nod al arborelui BST prin left, right(de tip pointeri catre NODE) si team(de tip pointer catre TEAM)

Structura AVLNODE: Defineste un nod al arborelui AVL prin left, right(de tip pointeri catre AVLNODE), team(de tip pointer catre TEAM) si height(de tip int, inaltimea nodului)

In main am apelat functiile pentru deschiderea fisierelor si citirea din acestea. Apoi, in functie de cerinta ce trebuia rezolvata, am apelat una dintre functiile Rezolva1, Rezolva2, ..., Rezolva5, ce au rolul de a rezolva cerinta data, respectiv de a elibera memoria alocata dinamic.

In functia de citire, am citit numarul de echipe, apoi am alocat dinamic spatiu pentru primul element al listei(santinela). Apoi am alocat spatiu pentru fiecare dintre cele n echipe, realizand si legaturile dintre acestea in lista initiala. Am citit numele echipei intr-un sir auxiliar, apoi am alocat dinamic spatiu pentru strlen(sir auxiliar) caractere. La fel am procedat si in cazul citirii numelui si prenumelui unui jucator. De asemenea, am citit dimensiunea echipei, apoi am alocat dinamic spatiu pentru un tablou de jucatori cu acea dimensiune.

In rezolvarea cerintei 1, am realizat o functie de afisare a echipelor citite si am apelat-o in functia de rezolvare a cerintei.

In rezolvarea cerintei 2, am realizat functia EliminaEchipe, care primeste ca parametri lista de echipe si numarul de echipe din aceasta. Prin functia EchipeRamase, am stabilit care este numarul de echipe ce trebuie sa ramana in lista, folosindu-ma de operatii pe biti. Operatia n = n&(n-1) elimina bitul cel mai nesemnificativ al reprezintarii binare a lui n. Este cunoscut faptul ca orice numar poate fi scris ca suma de puteri ale lui doi(exemplu: 75 = 2^6 + 2^3 + 2^1 + 2^0), iar fiecare bit cu valoarea 1 al unui numar pozitiv este 2 la o putere(pozitia bitului). Am folosit un while pentru a repeta operatia pana cand ramane un singur bit in reprezentarea lui n(adica n&(n-1) = 0, cu n!=0), acesta fiind bitul de pe pozitia cea mai indepartata, adica cea mai mare putere a lui 2 din numarul n(pentru exemplul de mai sus, 2^6). Am

memorat valoarea initiala a lui n in variabila nr si am eliminat echipe, dupa cum urmeaza, pana cand numarul de echipe devine n(cel calculat mai sus). Am realizat o functie care returneaza minimul experientei echipelor din lista si numarul de aparitii ale minimului. Am parcurs lista de nr - n ori, incepand de la variabila start(de tip *TEAM, care poate avea ca valoare fie capul listei, fie echipa la care s-a ramas la descoperirea unei valori minime ce apare de cel putin doua ori si nu este egala cu media primei echipe). La gasirea unei valori minime, aceasta este eliminata(eliberand memoria pentru toate elementele alocate dinamic), apoi se calculeaza noul minim, se stabileste pozitia de plecare in urmatoarea parcurgere a listei, si se intrerupe parcurgerea actuala a listei(break). Dupa ce este completata eliminarea, se foloseste functia de afisare a valorilor din lista.

Pentru a rezolva cerinta 3, se elimina echipele pana cand n este putere a lui 2, apoi este folosita functia AdaugaMeciuri care parcurge lista de echipe si grupeaza echipele doua cate doua formand un meci, care este pus in coada. Sunt folosite adresele echipelor din lista, astfel incat sa nu se aloce memorie inutil. Memoria este apoi eliberata pentru capul listei(santinela), toate informatiile necesare fiind acum stocate in coada de meciuri. Se afiseaza meciurile, dupa care se stabilesc castigatorii si invinsii(in functia CreeazaStive). Se adauga cate 1 punct la experienta fiecarui jucator din echipa castigatoare si la media echipei, echipa castigatoare este inserata in stiva de castigatori(topWinners), iar cea invinsa este inserata in stiva de invinsi(topLosers). De asemenea, nu se aloca momorie pentru noi structuri de tip TEAM, se folosesc cele alocate inca din citire. Se sterge stiva de invinsi, eliberand memoria alocata, dupa care se afiseaza castigatorii. Apoi se creeaza din nou coada de meciuri la fel ca la inceput, insa de data asta folosind informatiile din stiva de castigatori. Procesul se repeta pana cand ramane o singura echipa, care se va afla in stiva de castigatori. Aceasta se sterge, apoi este eliberata memoria.

In rezolvarea cerintei 4, se urmeaza pasii de la cerinta 3, pana cand raman exact 8 echipe. Pentru urmatoarele meciuri, echipele invinse sunt intai transformate in noduri ale arborelui BST prin functia CreeazaNod, dupa care sunt inserate in arbore prin functia AdaugaNod. Functia CreeazaNod aloca dinamic memorie pentru un nod al alborelui si informatiile memorate in acesta, apoi copiaza informatiile echipei si returneaza nodul astfel creat. Functia AdaugaNod cauta recursiv pozitia de inserare in arbore astfel incat acesta sa nu isi piarda proprietatea de BST, dupa care stabileste legaturile intre arbore si nod. De asemenea, daca punctajul echipei memorate in nod este egal cu punctajul altei echipe care deja se afla in arbore, se anuleaza inserarea si se elibereaza memoria alocata pentru nodul creat si pentru elementele incluse in acesta. Se repeta procesul pana cand ramane o singura echipa(cea castigatoare), care este si ea inserata in arbore. Apoi se apeleaza functia AfiseazaClasament, care afiseaza clasamentul parcurgand arborele BST inordine de la dreapta la stanga.

In rezolvarea cerintei 5 se modifica functiile de adaugare si creare, astfel incat sa lucreze cu noduri de tip AVLNODE in locul celor de tip NODE, memorand astfel si inaltimea pentru fiecare nod. In functia de adaugare, nodul se adauga recursiv, iar pe drumul invers din recursivitate se modifica(daca este cazul) inaltimile nodurilor de pe calea parcursa de nod pentru a fi inserat. Tot pe drumul invers, se stabileste si factorul de echilibru(diferenta dintre inaltimea fiului stang si inaltimea fiului drept al unui nod), iar daca modulul acestuia este egal cu 2, inseamna ca subarborele cu radacina in nodul respectiv nu are proprietatea de AVL si trebuie echilibrat. Sunt tratate toate cele 4 cazuri(LL, RR, LR, RL), fiind implementata cate o functie pentru fiecare. Se stabileste cazul in care se incadreaza nodul, apoi se apeleaza functia de rotire necesara(rotire la stanga; rotire la dreapta; rotire la stanga, apoi la dreapta; rotire la dreapta, apoi la stanga). Se repeta algoritmul pentru fiecare nod, apoi se afiseaza clasamentul care este acelasi ca in cazul arborelui BST(deoarece stim ca un arbore AVL este si BST). La final se

elibereaza memoria alocata nodurilor. Functia de afisare a nodurilor de la nivelul doi al arborelui functioneaza corespunzator, insa este pusa sub forma de comentariu in sursa.

La fiecare cerinta am eliberat memoria alocata si am inchis fisierele de citire si scriere. De asemenea, am tratat cazurile in care alocarea dinamica nu poate fi realizata, afisand un mesaj de eroare si oprind executia programului. Am verificat sursa folosind utilitarul valgrind pe fiecare test in parte, nefiind detectate erori sau memory leaks.