L5: Ansamblu

Să se implementeze în C++ o aplicație pentru rezolvarea unei probleme (indicate) folosind un **ansamblu** ca și *structură de date*. În cazul în care nu se precizează explicit, se va folosi ansamblu binar. Nu se vor folosi containere predefinite (din STL) pentru implementarea ansamblului.

Problemele se vor rezolva în echipe de 2 membri (decise la laborator). Fiecare membru al echipei va primi aceeași notă pentru tema primită.

Antetele operațiilor/interfețele necesare și testele pentru probleme se găsesc la http://www.cs.ubbcluj.ro/~gabis/sda/Laborator/TAD%20Interfete%20si%20teste/Interfete%20Laborator%205%20/

1. Se dau *k* liste ordonate în raport cu o relație de ordine $\mathfrak{R}$ între elemente. Se cere să se interclaseze cele *k* liste. Pentru reprezentarea listelor de intrare se va folosi clasa **list** din biblioteca STL.
2. Se dau *k* vectori ordonați în raport cu o relație de ordine $\mathfrak{R}$ între elemente. Se cere să se interclaseze cei *k* vectori. Pentru reprezentarea vectorilor de intrare se va folosi clasa **vector** din biblioteca STL.
3. Să se implementeze **TAD Coada cu priorități** folosind un *ansamblu ternar* (în loc de 2 descendenți, vor fi 3). Se va folosi o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).
4. Să se implementeze **TAD Coada cu priorități** folosind un *ansamblu cuaternar* (în loc de 2 descendenți, vor fi 4). Se va folosi o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).
5. Să se implementeze, folosind un ansamblu binar, un container **CP2** similar cu **Coada cu priorități**, exceptând faptul că vrem să accesăm și să ștergem **al doilea cel mai prioritar element** în raport cu o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).
6. Să se implementeze, folosind un ansamblu *ternar* (în loc de 2 descendenți, vor fi 3), un container **CP2** similar cu **Coada cu priorități**, exceptând faptul că vrem să accesăm și să ștergem **al doilea cel mai prioritar element** în raport cu o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).
7. Să se implementeze, folosind un ansamblu *cuaternar* (în loc de 2 descendenți, vor fi 4), un container **CP2** similar cu **Coada cu priorități**, exceptând faptul că vrem să accesăm și să

ștergem **al doilea cel mai prioritar element** în raport cu o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).

8. Să se implementeze, folosind un ansamblu binar, un container **CP3** similar cu **Coada cu priorități**, exceptând faptul că vrem să accesăm și să ștergem **al treilea cel mai prioritar element** în raport cu o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**).
9. Să se implementeze, folosind un ansamblu binar, un container **CP*k*** similar cu **Coada cu priorități**, exceptând faptul că vrem să accesăm și să ștergem **al *k*-lea cel mai prioritar element** în raport cu o relație de ordine $\mathfrak{R}$ între priorități (dacă $\mathfrak{R}=\leq$, atunci elementul cel mai prioritar este **minimul**). ***Indicație*: *pentru determinarea elementului cel mai prioritar dintre k elemente, se va folosi tot un ansamblu binar*.**
10. Se consideră un vector conținând *n* numerele naturale distincte. Să se calculeze suma celor mai mari *k* (*k*>0) numere din vector, folosind un algoritm de complexitate $O(n \cdot \log_2 k)$. Pentru reprezentarea vectorilor de intrare se va folosi clasa **vector** din biblioteca STL.
11. Se consideră o listă conținând *n* numerele naturale distincte. Să se șteargă cele mai mici *k* (*k*>0) numere din listă, folosind un algoritm de complexitate $O(n \cdot \log_2 k)$. Pentru reprezentarea listei se va folosi clasa **list** din biblioteca STL.
12. Se dă un vector *v* cu elemente de tip **TComparabil**. Se cere să se construiască un vector format din primele *k* (*k*>0) (cele mai *mici*) elemente din *v* în raport cu o relație de ordine $\mathfrak{R}$ între elemente (dacă $\mathfrak{R}=\leq$, atunci cel mai *mic* element se va considera **minimul**). Se va folosi un *ansamblu ternar* (în loc de 2 descendenți, vor fi 3). Pentru reprezentarea vectorului se va folosi clasa **vector** din biblioteca STL.
13. Se dă un vector cu elemente de tip **TComparabil**. Se cere să se șteargă ultimele *k* (*k*>0) (cele mai *mari*) elemente din vector în raport cu o relație de ordine $\mathfrak{R}$ între elemente (dacă $\mathfrak{R}=\leq$, atunci cel mai *mare* element se va considera **maximul**). Se va folosi un *ansamblu cuaternar* (în loc de 2 descendenți, vor fi 4). Pentru reprezentarea vectorului de intrare se va folosi clasa **vector** din biblioteca STL. Nu se vor folosi operații de sortare.
14. Se dă un vector cu elemente de tip **TComparabil**. Se cere să se determine produsul primelor *k* (*k*>0) cele mai mari elemente din vector. Se va folosi un *ansamblu n-ar* (în loc de 2 descendenți, vor fi *n*). Pentru reprezentarea vectorului se va folosi clasa **vector** din biblioteca STL.