**SD**

**Variabile statice** - vizibile oriunde in program, iar daca sunt declarate intr-o functie si modificate, sunt vizibile si in urmatorul apel.

**Funcții statice** - restricționeaza accesul la ea doar la fișierul în care este declarată.

**Const** - variabilă constantă, read-only, a cărei valoare nu se va schimba.

**Alocarea statică** se face în zona de date (variabilele globale și statice - la încărcarea programului în memorie) și **alocarea dinamică** se face pe heap (la rularea programului).

**Lista** nu are elementele intr-o zona contigua ca **vectorul**, ci foloseste toată memoria liberă disponibilă, indiferent de dispunerea ei.

**Hashtable** - mapeaza cheile la valori (un array de liste, câte o listă înlănțuită pentru fiecare bucket)

- cautare cu **liste**: O(n), cu **ABC**: O(log n).

**Graf** - este o pereche de mulţimi G = (V, E)

- liste de adiacenţă (Adj) pt grafuri rare sau matrice de adiacenţă.

**BFS** - o coadă (**Q**) cu nodul sursă. Se vizitează copiii nodului şi se pun în coada. Daca nu mai există vecini nevizitați, nodul sursă este scos din coadă.

- folosit pt **drumul minim**

- se obtine arbore de acoperire (BFS din fiecare CC)

- o componenta conexa este cea din care este nod sursa, daca dupa mai sunt noduri nevizitate, se aplica bfs pe fiecare.

**DFS** - o stivă (**S**), pt nodul din vârful stivei se adauga primul copil nevizitat si tot asa, iar daca nu mai sunt nevizitati se scoate de pe stivă.

- fiecare nod are două momente de timp **tDesc[u]** (descoperire nod) si **tFin[u]** (încheiere procesare nod)

**Complexitate** BFS, DFS: **O(|E|+|V|)** - pt Adj, **O(|V|^2)** - pt matrice.

**Algoritmul Floyd-Warshall** - căutarea celor mai scurte căi în graf orientat cu costuri.

**Componente conexe** (neorientat) - submultime maximala de noduri si există un lanţ de la A la B si de la B la A (se gasesc cu BFS, DFS).

**Componente tare conexe** (orientat) – CC, doar ca in graf orientat.

**TopSort** - ordonare liniară a vârfurilor astfel încât, pt (u,v), u trebuie să apară **înaintea** lui v

- parcurgere DFS, sortare descrescatoare dupa tFin.

**Graf hamiltonian** - are un **ciclu** hamiltonian, adica trece prin fiecare nod o singură dată si dacă **nod început == nod sfârşit**.

**Arbore binar** - un graf neorientat conex aciclic, implementat cu vector/structura cu pointeri

- **complet**: toate nivelurile sunt pline, mai putin ultimul (completat stanga-dreapta)

- **preordine**: RSD

- **inordine**: SRD

- **postordine**: SDR

**ABC** - cheia dintr-un nod este mai **mare** decât cheile nodurilor din subarborele **stâng** și mai **mică** decât cheile nodurilor din subarborele **drept.**

- complexitate: **O(log n)**

- parcurgere in inordine: sortare crescatoare

**minHeap** - un arbore binar în care fiecare nod are valoarea mai mare sau egală cu cea a părintelui său.

- se salveaza ca vector, mereu **echilibrat**

- folosit pentru găsirea celui de-al k-lea cel mai mic/mare element

**PA**

**MergeSort:** imparte in **2 subvectori** de **n / 2**, ii sorteaza prin interclasare si la final ii interclaseaza –> O(n \* log n).

**BinarySearch:** se imparte vectorul (sortat) in **subvectori** de **n / 2** si se cauta in cel potrivit –> O(log n).

**BubbleSort:** clasic, 2 foruri –> O(n^2).

**Dijkstra:** drumul **minim** de la sursa la toate nodurile, doar cu costuri pozitive –> O(m \* log n).

**Bellman-Ford:** drumul **minim** de la sursa la toate nodurile, cu costuri oarecare –> O(n \* m).

**Roy-Floyd:** drumul **minim** intre **oricare 2** noduri (prin u-k-v ajunge mai repede decat u-v) –> O(n^3).

**Arbore acoperire:** subgraf cu nr minim de **muchii** ce interconecteaza **toate nodurile**.

**CN**

**SRAM:** volatila, scumpa, rapida si mica, folosit pt **Cache**.

**DRAM:** volatila, mai ieftin, lent si mare, trebuie **refresh**, folosit pt **Main**.

**Cache:** memorie SRAM rapida, mai mica, intre **procesor** si **memoria principala** pt a stoca date accesate **frecvent** (aproape de viteza procesorului).

Memorie **nevolatila**: ROM, Flash

**Latenta:** timp/intarziere pt un acces la memorie.

**Latime banda:** nr accese / timp.

- localitate **temporala**: daca este accesata o locatie, probabil ca **aceeasi** locatie va mai fi accesata

- localitate **spatiala**: daca este accesata o locatie, probabil sa fie accesate si cele **vecine**

**Arhitectura von Neumann**: **o singura** memorie pt instructiuni si date (**aceeasi**).

**Arhitectura** **Harvard**: 1 memorie pt instructiuni si 1 memorie pt date (**separate**).

**Memoria virtuala:** mapata intre **RAM** si **hard-drive**, creeaza iluzia unei memorii mai mari

- **programele** folosesc adrese virtuale, iar **CPU** converteste adresele virtuale in fizice

- Main (DRAM) este ca un **cache** pt hard-disk

**PCom**

**ISO-OSI:** fizic, legatura date, retea, transport, sesiune, prezentare, aplicatie

**TCP/IP:** fizic, retea, transport, aplicatie

**Detectare erori:** suma de control (CRC) si o punem la finalul header.

**- fizic:** transmiterea sirurilor de biti pe un canal.

**- legatura date:** comunicatia intre 2 dispozitive conectate direct + adauga MAC.

**- retea:** trimiterea pachetelor prin retele intermediare (rutarea si adresarea) (IP, ICMP, ARP) + adauga IP sursa si destinatie.

**- transport:** asigura transferul datelor corect si eficient (UDP, TCP) + adauga port iesire.

**- aplicatie:** DNS, SMTP, POP3, FTP

**Ruterul:** daca pachetul e in aceeasi retea il livreaza, daca nu trimite mai departe conform tabelei de rutare (sau la default gateway).

**ARP:** afla o adresa MAC, stiind adresa IP (se face broadcast **ARP request** cu adresa IP destinatie si raspunde cu MAC-ul lui doar cel cu adresa cautata (daca e in **aceeasi** retea) sau ruterul raspunde cu MAC-ul lui (daca **nu** este in retea, **Proxy ARP**).

**NAT:** translateaza o adresa **privata** in una **publica** si ca sa poate comunica interiorul cu exteriorul.

**IP:** utilizează adrese IP pentru a **identifica fiecare dispozitiv** în rețea și pentru a ruta.

**ICMP**: identifica erorile aparute in retea (trimite mesaje de eroare și de corectie către dispozitive).

**UDP**: simplu, mai rapid, **nu** trebuie conexiune prestabilita, livreaza datagrame "best-effort" (se pot pierde/ajunge in alta ordine), **nu** face controlul fluxului.

**TCP**: foarte folosit, livreaza **tot**, in ordine, livrare **sigura** pe retea **nesigura** (are ack), **trebuie** conexiune presabilita.

**DNS:** leaga o adresa **IP** la un **nume de domeniu**, se fac cereri DNS.

**SMTP:** comunicatia intre serverele de email; cu el ajunge mail-ul nostru la server.

**VPN:** retele private peste Internet; la fiecare oficiu este un firewall si se fac **tunele** intre firewall-uri.

**Browser:** determina **URL**, cere **DNS**-ului adresa **IP** pt [www.xxx.yyy](http://www.xxx.yyy) si acesta ii raspunde. Deschide o conexiune **TCP** pe port 80 (http) pe acel IP, trimite **GET** cu ce vrea sa obtina si **serverul trimite** fisierul dorit. Se inchide conexiunea tcp si se **afiseaza** continutul fisierului.

**URL:** schema de **adresare** a resursele de pe Internet

**HTTP:** transportul mesajelor intre clienti si server; accesarea informatiilor in Internet

* **GET:** cerere de **citire** a unei **pagini** Web + primire **raspuns**
* **HEAD:** ca get, dar returneaza **antetul** unei pagini Web
* **POST:** trimite date la server pt a crea/actualiza

**1xx** – informatie, **2xx** – succes, **3xx** – redirectare, **4xx** – eroare la client, **5xx** – eroare la server

**RL**

**Tabela CAM:** asociere **porturi** – adresa **MAC** (pe switch)

- **comutarea**: se parcurge secvential CAM si cauta asocierea lui MAC destinatie (daca nu exista, flooding)

- se populeaza **dinamic** (adauga MAC sursa cand trece pachetul prin switch)

**Adresa retea**: AND intre masca si adresa IP

**Adresa broadcast**: OR intre masca inversata si adresa IP

**Tabela ARP**: asocieri **IP – MAC**

**VLAN** - **access** port: apartine unui singur VLAN

- **trunk**: pot merge mai multe VLAN-uri pe aceeasi legatura

**STP:** inchide unele porturi pt a **nu** avea **bucle** (pe o retea de switch-uri).

**Tabela de rutare:** are rutele cele mai eficiente, ordonate masca mare –> mica.

**Firewall:** blocheaza traficul nedorit din retea (pe ruter/dispozitiv dedicat de firewall).

**Iptables:** **filtrare** pachete (FILTER), **translatare** adrese (NAT), **rescrie** antetele pachetelor (MANGLE).

**Adrese private:** nu pot fi folosite in Internet => trebuie translatare.

**Atac de recunoastere**: aflarea informatiilor (IP statiilor din retea, serviciile de pe statii).

**Atac DDoS:** se trimit **multe cereri** pt a impiedica procesarea normala (nu se stie care sunt cereri valide si care atac).

**Atac de acces:** aflarea parolelor.

**APD**

**Race condition:** 2 thread-uri executa **aceeasi** secventa, dar sunt **rezultate diferite.**

**Mutex:** mecanism sincronizare pentru a evita conflictul între thread-uri care încearcă să acceseze **aceeași resursă** în **același timp**.

**Semafor:** lasa doar un nr de thread-uri sa intre in zona critica.

**Bariera:** merge dupa bariera dupa ce au ajuns toate thread-urile la ea.

**Deadlock:** thread-urile se **blocheaza** la infinit, ca vor sa acceseze **aceleasi resurse ocupate** de **alt thread** in acelasi timp (Producator-Consumator, Filosofi).

**Livelock:** un thread raspunde simetric la actiunea celuilalt (dar nu sunt blocate).

**Volatile:** **nu** se face **caching**, citeste valoarea direct din RAM.

**POO**

- **final**  – **variabila**: o singură inițializare (immutable)

– **clasa/metoda**: **nu** pot fi mostenite/suprascrise

- **static:** camp/metoda ce **nu** depinde de o **instanta** (se acceseaza cu NumeClasa.nume) (are o zona proprie in memorie)

Principii:

**1. Abstractizare:** “ascunde” unele functionalitati sa **nu fie vizibile in exterior** (ex: nu toti trebuie sa stie cum se face salvarea in BD)

- **clasa abstracta**: nu are instanta, **nu** trebuie implementate **toate metodele**

- **interfata:** metode fara implementare, la “implements” **trebuie** implementate **toate** metodele

**2. Incapsulare**: gruparea datelor si metodelor intr-o clasa, ascund/restrictioneaza implementarea, iar accesul se face prin metode publice (ex: Sa nu se poata schimba username/parola fara sa aiba voie) – specificatori acces

**3. Mostenire:** preia functionalitati dintr-o clasa existenta, mai **adauga** unele noi sau le **modeleaza** pe cele existente (implementare specifica a clasei generale, ex: clasa Vehicle –> clasele Car, Truck)

**4. Polimorfism:** abilitatea unei clase **să se comporte ca o altă clasă:**

- **Overriding** dinamic (run-time): suprascrie metodele din clasa parinte pt a le **specializa** (trebuie acelasi return si semnatura, alegerea se face pe baza tipului obiectului = instanța)

- **Overloading** static (compile): mai multe metode cu **același nume**, dar implementari, semnaturi diferite (compilatorul decide ce metodă este apelată pe baza tipului referinței, parametrilor)

**Singleton** - se creeaza **doar o instanta** a unui obiect (lazy – se instantiaza cand e nevoie, eager – la inceput).

Visitor - separa metode de obiectele pe care le opereaza si le pune intr o clasa

**Factory** - implementeaza **aceeasi interfata**/clasa abstracta

- **izoleaza** un obiect ce are nevoie de o instanta, de crearea acesteia

**Observer** - organizarea comunicatiei intre clase in functie de **scopurile** lor

- subiectul trimite **notificari** observatorilor

**Strategy** - **incapsuleaza** algoritmii intr-o clasa si la runtime sunt alesi cei necesari

- se alege **strategia** in functie de **situatie** (PayPal sau Card)

**Command** - **decupleaza** obiectele care executa actiuni de cele care le invoca

- folosit pt undo, logging

**Builder** - elimina **supraincarcarea** constructorilor

- creare **flexibila** a obiectelor