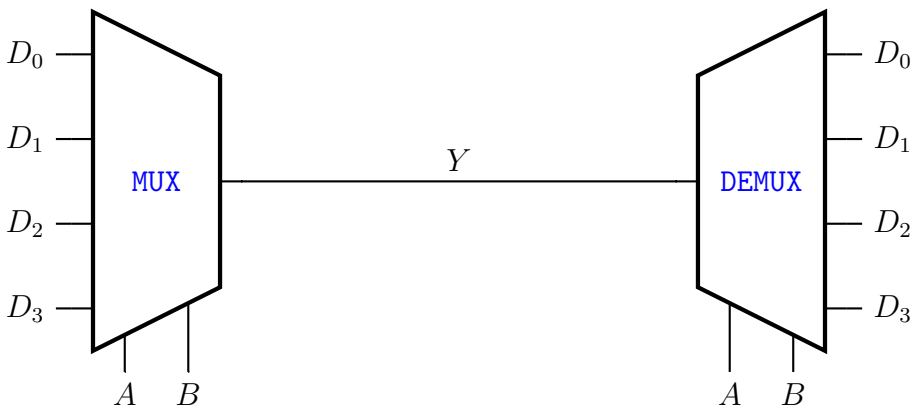
Tartalomjegyzék



1. Tétel – Az objektum-orientált programozás alapjai

Objektum

Az objektum (`object`) egy rendszer egyedileg azonosítható szereplője adatokkal és működéssel.

Osztály

Az osztály (`class`) megegyező szerkezetű, hasonló viselkedésű elemek mintája. Ez egy felhasználó által definiált típus. Az osztályhoz tartozó példány (`instance`) az objektum. Adatszerkezetből és tagfüggvényekből áll. Alaphelyzetben az adattagok és a tagfüggvények nem láthatóak. (`data hiding`)

Struktúra

A struktúrák (`struct`) adattagjai kívülről alaphelyzetben elérhetőek, ellentétben az osztályokkal.

Tagfüggvények

Az adattípussal kapcsolatban álló (az adattagokon működő) függvények lehetnek belül, vagy kívül definiáltak. Lehetnek továbbá `inline` függvények. (Hívás helyén lesznek lefordítva.)

Konstruktor

Függvény az objektumok előkészítésére, esetleg inicializálása. Ugyanaz a neve mint az osztálynak, nincs típusa, sem visszatérési értéke. Létrehozás után nem hívható. Ha nem adjuk meg akkor is létezik alapértelmezett. (`default`) Másoló konstruktor megadása konstans referenciával lehetséges.

Destruktor

Az objektumok megszűnésekor automatikusan meghívódó függvény. A neve: `~ClassName`. Nincsenek paraméterei. Nem terhelhető túl. Statikus `instance` esetén a `scope`-on kívül, dinamikus esetben a `delete` kulcsszó esetén hívódik meg.

Statikus tagok

A statikus (`static`) tagok nem az objektumhoz, hanem az osztályhoz vannak kötve. Az osztályban nem definiálható, névtér (`namespace`) szintjén kell ezt megtenni.

Friend függvények

A `friend` függvények segítségével az osztály rejtett tagjai is elérhetőek.

2. Tétel – Az objektum-orientált programozás alapjai

Operátorok túlterhelése

Szinte minden operátor túlterhelhető, kivéve: `::`, `.`, `.*`, `?`, `sizeof()`, `typeid()`. Új műveleti jel nem hozhető létre. Egyoperandusú művelet esetén:

- `opetaror++();` → `myVar++` (Utótag)
- `opetaror++(int);` → `++myVar` (Előtag)

C++ IO

Az `std namespace`-t használjuk, az alábbi fejlécekkel: `iostream`, `iomanip`.

```
#include <iostream>
#include <iomanip>

cin.get();
cout.put('\n');

cout << "Hello World"
string tmp;
cin >> tmp;

cout.flags(ios_base::hex)
  // (no)boolalpha
  // left, right
  // dec, hex, oct
  // fixed, scientific
cout.precision(2)

cout << setw(5) << setprecision(2) << 12.345;

// Overloading
friend ostream& operator<< (ostream& os, const type& myVar);
friend istream& operator>> (istream& is, type& myVar);

```

New, delete

A `new` és `delete` kulcsszavak segítségével dinamikus példányokat hozhatunk létre, és törölhetünk. Ezek is túlterhelhetőek:

```
void* operator new(size_t size){
  // { ... }
  return new type[size];
}

void operator delete (void* p, size_t size){
  // { ... }
  ::delete p;
}

```

Osztály hierarchiák

Az újrahasznosítható szoftver alapja. Többszörös öröklődés (`inheritance`) lehetséges.

3. Tétel – Az objektum-orientált programozás alapjai

Öröklődés

	public	protected	private
public	public	protected	-
protected	protected	protected	-
private	private	private	-

- osztály más osztályok tulajdonságait/viselkedését is magába integrálja
- módosított viselkedésű osztály az eredeti kód másolata, hivatkozása nélkül
- minden változás automatikusan végigmegy a hierarchián
- az ős neve a bázis osztály (`base/parent class`)
- az utód neve leszármazott osztály (`derived/child class`)
- konstruktor, destruktor, barát függvények, illetve az `operator=` túlterhelése nem öröklődnek, öröklődnek viszont az adattagok, tagfüggvények és a többi túlterhelt operátor

Egységbe zárás

Az objektum egységbe zárja (`encapsulation`) az adatokat és a programokat. Magába foglalja a külvilág felé mutatott viselkedést. A belső struktúrája, állapota, adatai és kezelőfüggvényei kívülről nem láthatóak. (`data hiding`)

Protected osztálytagok

A `protected` tagváltozók öröklődés esetén nyilvánosak az utódosztály számára, de kívülről nem elérhetőek.

Kompozíció, aggregáció

Kompozíció esetén egy meglévő osztályt tagobjektumként használunk fel egy másik osztályban. (statikus példány) Aggregáció esetén pointert vagy referenciát használunk. (dinamikus példány) A különbség akkor lép fek, ha a felhasznált osztályt módosítani kezdjük.

Többszörös öröklődés

Többszörös öröklődés esetén több ős van.

Barátság

Az alaposztály barátja az utódban az öröklött tagokat éri el. Az utód barátja az ős `public` és `protected` tagjait éri el.

```
1 friend type myFunc();            // Kulso fuggveny
2 friend type MyClass::myFunc();   // Masik osztaly publikus tagfuggvenye
3 friend MyClass;                  // Masik osztaly osszes fuggvenye
4
```

4. Tétel – Az objektum-orientált programozás alapjai

Polimorfizmus

A szó jelentése: ugyanaz a metódus más- és másképpen működik a családfa szintjein. A tagfüggvények újradefiniálhatóak. Lehet a fordításkor korai, statikus kötés, vagy futás közben késői, vagy dinamikus kötés. Ősosztály-típusú mutatóval meghívható az ős metódusa a leszármazottra.

```
class Parent{};
class Child : public Parent{
  public:
    Child() : Parent() {};
};

```

Virtuális alaposztályok.

A `virtual` kulcsszót használva a tagfüggvények átdefiniálhatóak lesznek a leszármazottakban. "`=0`" esetén tisztán virtuális lesz.

```
virtual void myFunc = 0;

```

A virtuális függvények virtuálisak maradnak, a származtatott osztályok hierachiájában újra definiálhatjuk őket a `virtual` kulcsszó nélkül. (`C++11` óta) Újradefiniáláskor pontosan egyező függvényt kell megadni (név, típus, paraméterlista), egyébként túlterhelés lesz. A futás közben dől el, hogy melyik tagfüggvény aktivizálódik. Az aktiválás az ős pointerén vagy referenciáján keresztül egy a virtuális függvények címeit tartalmazó táblázat (`VMT`) segítségével.

Abstract osztály

Tisztán virtuális függvényt tartalmazó osztály nem példányosítható. Ezeket `abstract` osztályoknak hívjuk. Az `interface` olyan absztrakt osztály, amely csak tisztán virtuális függvényeket tartalmaz.

Általánosított osztályok

Általánosított osztályokat `template` segítségével készíthetünk.

```
template<typename T>
class ClassName{
  T myFunc (T myVar){
    // { ... }
    return myVar
  }
};

```

A `template` minden használó forrás állományban kell.

5. Tétel – Az objektum-orientált programozás alapjai

Standard Template Library

Adatstruktúrákat és algoritmusokat tartalmaz C++ nyelvhez.

Tárolók

Soros és asszociatív tárolókat tartalmaz az STL. A soros tárolók jellemzője, hogy megőrzik az elemek beviteli sorrendjét. Az asszociatív tárolók elemei egy kulcs alapján érhetők el. Lehetnek rendezettek, illetve rendezetlenek. Mindegyiknek van default és copy konstruktora, = operátora.

Bejárók

A bejárók (iterator) pointer szerű, tároló független objektumok. Teljes bejárás esetén a begin()/rbegin() és end()/rend() használatos. Egy pozíciót határoznak meg a tárolóban. Iteratorokra alkalmazható függvénysablonok:

```
1  begin(); end();
2  advance(<int>count);
3  distance(iterator1, iterator2);
4  next(iterator, <int>count=1);
5  prev(iterator, <int>count=1);
6
```

Algoritmusok

Algoritmusok a konténerek adatainak kezelésére (sorbarakás, keresés, szélsőértékek ...)

Függvényobjektumok

Függvényobjektumok megvalósítják a függvényhívás () operátorát az algoritmusokban. Hívásakor az osztály objektumát adjuk át.

```
1  class Even{
2    public: bool operator() (int i) const {
3      return (x%2) == 0;
4    }
5  };
6  vector<int> numbers{2, 3, 6, 9};
7  vector<int>::iterator i = find_if(
8    begin(numbers), end(numbers), Even()
9  );
10
```

C++11 óta általánosított függvényobjektumok is elérhetőek. Szintén ettől a szabványtól használatosak a névtelen (lambda) függvények.

- `[]{};` – nem használ a hívó hatókörében változókat
- `[=]{};` – a hívó hatókörében definiált változók másolatát látja
- `[&]{};` – a hívó hatókörében definiált változók referenciát látja
- `[&,i]{};` – a hívó hatókörében definiált változók referenciát, `i` másolatát látja
- `[=,&i]{};` – a hívó hatókörében definiált változók másolatát, `i` referenciát látja

Soros tárolók

Az array adott méretű és típusú tárolótömb folytonos memóriaterületen. for ciklussal bejárható.

```
// Array
#include <array>

array<type, size> myArr { {myVar1, myVar2, myVar3} };

myArr[0] = 12; // feltoltes [] operatorral, bejarhato for ciklussal

front(); back(), data(); at(); []          // elem eleres
begin(); end(); rbegin(); rend();          // iterator
empty(); size(); max_size()                // meretek
swap(); fill(); reverse(); accumulate();   // muveletek

```

Az allokátorobjectumok (allocator) a tárolók számára lefoglalt memóriaterületet kezelik.

```
// Allocator
#include <memory>

allocator<type> myAlloc;

allocate(<size_t> n);
construct(<pointer> p, <const reference> r);
deallocate(<pointer> p, <size_t> n);
destroy();

```

A vector dinamikus tömbökkel megvalósított soros tároló. Folytonos adatterületen pointerrel és bejáróval is bejárható. Automatikusan növekszik és csökken a tárolási mérete. Az elemek könnyen elérhetők pozíció alapján. (állandó idővel) Elemek sorban bejárhatóak (lineáris idővel) Elemek illeszthetők/törölhetők a végéről. (konstans idővel) Elemek be is illeszthetőek/törölhetőek, de erre más tárolók (deque, list) jobb időt produkálnak.

```
// Vector
#include <vector>

vector<type> myVector1 (count, myVar);
vector<type> myVector2 (myVector1.begin(), myVector1.end());
vector<type> myVector3 (myVector1);

begin(); end(); rbegin(); rend();    // bejarok
[]; at(); front(); back();           // tulajdonsagok
size(); resize();                    // meretek (ossz)
capacity(), reverse();               // meretek (hatralevo)
push_back(), pop_back();             // modositok
iterator insert();                   // beszuras (nem hatekony)

// Bejaras
for(i = myVec.begin(); i != myVec.end(); i++);
for(int i = 0; i < myVec.size(); i++);
for(auto i : myVec);
for(auto& i : myVec);

```

A string is STL tároló.

```
// string
#include <string>

```

```
at(); []; front(); back(); data();
begin(); end(); rbegin(); rend();
empty(); size(), max_size();
push_back(); copy(); c_str();
find(); find_first(); insert(); replace();

```

A deque egy kétvégű sor, amely mindkét végén növelhető. Konstans idő alatt adhatunk hozzá, illetve távolíthatunk el elemet a sor végeiről. Nem folytonos adatterületen helyezkedik el. (nincs capacity() és reverse()) Egydimenziós tömböt tartalmazó listában tárolódik. Az elemek könnyen elérhetőek pozíció alapján. (állandó idővel) ([], at(), front(), back()) Elemek sorban bejárhatóak. (lineáris idővel) Lassabb, mint a sor. (queue)

```
// deque
#include <deque>

deque<type> myDQ1(count, myVar);
deque<type> myDQ2{myVar1, myVar2, ...};
deque<type> myDQ3(myDQ1.begin(), myDQ1.end());
deque<type> myDQ4(myDQ1);

// Konstans idovel elem hozzaadasa
push_front(); pop_front();
push_back(); pop_back();

```

A listák (list) esetén nincsen direkt elérés. (at(), []) Kétirányban láncolt lista. Gyors beszúrás törlés, sorbarakás összefésülés.

```
// list
#include <list>

front(); back(); push_front(); push_back(); pop_front(); pop_back();
iterator insert();
sort(); merge(); splice();

```

Az egy irányban láncolt listák (forward_list) esetén nincsen push_back() és pop_back(), viszont van insert_after().

```
// forward_list
#include <forward_list>

```

Asszociatív tárolók

Az asszociatív tárolók (map, set) absztrakt adattípusok. A set egyetlen elem sort tartalmaz, ami a kulcs egyben. Nincsen [] operátor.

```
// set
#include <set>

type array[] = {var1, var2, var3}
set<type> mySet1(array, array+3)
set<type> mySet2(mySet1.begin(), mySet1.end())
set<type> mySet3(mySet1)

begin(); end(); rbegin(); rend();
::iterator; size(); max_size(); insert();
empty(); erase(); clear(); find();

```

A map kulcs–érték adatpárokat tartalmaz. (a pair sablon szerint) Lehet definiálni az összehasonlítást kulcsra és értékre is. Van [] operátor. (kulcs kell)

```
// map
#include<map>

map<type1, type2> myMap1;
map<type1, type2> myMap2(myMap1.begin(), myMap1.end());
map<type1, type2> myMap3(myMap1);

begin(); end(); rbegin(); rend();
::iterator; size(); max_size(); insert();
empty(); erase(); clear(); find();

```

Léteznek kulcsismétlést megengedő változatok (multiset, multimap), a keresés ebben az esetben lineáris végrehajtási idejű. Mindegyiknek van rendezetlen változata is. (unordered_xxx)

Konténer adapterek

A verem (stack) FILO típusú tároló. Interface minden standardnak, aminek van back(), push_back() és pop_back() művelete. (vector, deque, list)

```
// stack
#include <stack>

stack<type, container<type>>;

empty(); push(); pop(); top(); size()

```

A sor (queue) FIFO típusú tároló. Interface minden standardnak, aminek van back(), push_back() és pop_back() művelete. (vector, deque, list)

```
// queue
#include <queue>

queue<type, container<type>>;

empty(); push(); pop(); front(); back(); size();

```

A prioritásos sor (priotity_queue) interface minden standardnak, aminek van back(), push_back() és pop_back() művelete. (vector, deque, list)

```
// queue
#include <queue>

queue<type, container<type>>;

empty(); push(); pop(); front(); size();

```

6. Tétel – A számítástudomány alapjai

Turing gép

- külső adat és tárolóterület: végtelen szalag, amelynek egymás után cellái vannak, amelyek vagy üresek, vagy jelöltek
- a gép egyszerre egy cellával foglalkozik (Az író/olvasó feje egy cellán áll)
- a szalagon tud jobbra-balra lépni, tud jelet olvasni, törölni és írni
- a bevitel, a számítás és a kivitel minden konkrét esetben véges marad, ezen túl a szalag üres
- a gép belső állapotait számozzuk
- a gép működését megadja egy explicit helyettesítési táblázat
- ha egy algoritmus elég mechanikus és világos, akkor található olyan Turing-gép, amely azt végrehajtja
- a Turing gép definiálja mindazt, amit matematikailag algoritmikus eljárás alatt értünk
- minden más algoritmikus eljárást végrehajtó rendszer ekvivalens valamelyik Turing-géppel
- megállási probléma – nem tudjuk, hogy egy adott programmal megáll-e
- nincs arra bizonyítási módszerünk, hogy egy eljárás biztosan algoritmus

Eljárások

Nem garantálható, hogy véges lépésben, tehát valaha is választ kapjunk kérdésünkre.

Algoritmusok

A választ véges számú lépés után mindenképpen megkapjuk. Egy véges utasítássorozat, amely bármely input esetén véges lépésszám után megáll, eredményt ad. Minden algoritmus leírható az alábbi logikai struktúrákkal:

- rákövetkeztetés (konkatenáció)
- választás (alternáció)
- ciklus (iteráció)

Maga a Turing gép matematikai leírása az algoritmus fogalmának formális definíciója. Minden probléma, amelyre eljárás, procedúra szerkeszthető, Turing-géppel megoldható. Az ember azokra és csakis azokra a kérdésekre tud választ adni, amelyekre a Turing-gép is képes. A tartalmazás kérdése algoritmikusan eldönthetetlen.

- bejárás – elemek keresése
- keresés – adott értéknek megfelelő elemek kiválasztása
- beszúrás – új adat beillesztése
- törlés – adatelem eltávolítása
- rendezés – elemeket logikai sorrendbe
- összeválogatás – különböző rendezett adathalmazokból új elemhalmaz kialakítása

7. Tétel – Adatszerkezetek

- egyszerű vagy összetett alapadatok rendszerének matematikai, logikai modelljе
- elég jó ahhoz, hogy tükrözze a valós kapcsolatokat
- elég egyszerű a kezeléshez

Tömbök

- lineáris, egy vagy többdimenziós
- n darab azonos típusú adatelemből áll
- az elemekre indexhalmazzal hivatkozunk
- az elemeket egymást követő memóriahelyek tárolják
- az elemekhez bejárás nélkül férünk hozzá

Kapcsolt listák

- a kapcsolt lista vagy egyirányú lista adatelemek, vagy csomópontok lineáris gyűjteménye, ahol az elemek sorrendjét rögzítjük
  - a mutatókat tároló elemeket kapcsolómezőnek hívjuk
- a kétirányú listák mindkét irányban bejárhatóak
- a ciklikusan kapcsolt listák nem rendelkeznek első és utolsó elemmel, hiszen az "első" az "utolsóra" mutat

Gráf

- A gráf két halmazzal jellemezhető adatszerkezet
  - a csomópontok sorszámozott halmazzal
  - az elemeket összekötő számpárral jellemzett élek halmaza
- az összekötött csomópontokat szomszédoknak hívjuk
- $\deg(u)$ a csomópont foka a befutó élek száma
  - ha $\deg(u) = 0$, akkor a csomópont izolált
- a v_0-ból v_n-be mutató élek halmazát útnak nevezzük – $P(v_0; v_1; \ldots; v_n)$
  - az út zárt, ha $v_0 = v_n$
  - az út egyszerű, ha minden pontja különbözik
  - az út kör, ha legalább 3 hosszú, egyszerű
- egy G gráf összefüggő, ha bármely 2 pontja között létezik egyszerű út
- egy G gráf teljes, ha minden csomópontja minden csomópontjával össze van kötve.
- egy G gráf címkézett, ha éleihez adatokat rendelünk

  - egy G gráf súlyozott, ha éleihez rendelt adatok nemnegatívak
- egy G gráf irányított, ha az éleknek irányítottságuk van
- rendelhetünk hozzájuk mátrixokat:
  - szomszédsági mátrix – $a_{ij} = 1$, ha i-ből j felé halad él
  - útmátrix – $a_{ij} = 1$, ha i-ből j-be halad valamilyen út

Fa

- a fa köröket nem tartalmazó összefüggő gráf
- a bináris fa elemek véges halmaza, mely:
  - vagy üres
  - vagy egyetlen **T** elemhez (gyökér) kapcsolt két diszjunkt **T1** és **T2** részfa alkotja (szukcesszor)
  - a zárócsomópontnak nincsen szukcesszora (levél), az utolsó élet ágnak nevezzük
  - egy generációba az azonos szintszámú elemek tartoznak (gyökér szintszáma 0)
  - a mélység az azonos ágon elhelyezkedő elemek maximális száma
  - teljes, ha az utolsó szintet kivéve a csomópontok száma maximális
  - ábrázolhatóak:
    - kapcsolt szerkezettel
    - tömbökkel
    - szekvenciálisan ($2k$ helyek eltolva)
  - bejárás történhet pl. irányítás szerint
- általános fa esetén nem csak 2 szukcesszor engedélyezett
  - elemek véges halmaza (**T**), amely ...
    - tartalmaz egy kitüntetett **R** gyökérelemet
    - a többi elem nem nulla diszjunkt részfája **T**-nek
- minimális feszítőfa probléma (minimum spanning tree)
  - minden csúcsot érintő, összefüggő, körmentes élhalmaz
  - `bemenet:` összefüggő, súlyozott, irányítatlan $G = (V; E)$ gráf, ahol $k(u; v)$ a súly, az $(u; v)$ az élköltséget fejezi ki
  - `kimenet:` egy **F** feszítőfa, melyre az élköltség minimális: $k(\mathbf{F}) = \sum_{(u;v)\in\mathbf{F}} k(u; v)$
  - megoldás Kruskal algoritmussal, amely egy mohó algoritmus
    - minden pillanatban a leghatékonyabb megoldást válassza ki

Verem

- a verem (stack) LIFO típusú tároló.
- push és pop függvényekkel rendelkezik

Sor

- a sor (queue) FIFO típusú tároló.
- push és pop függvényekkel rendelkezik

8. Tétel – Algoritmusok

Bejárás

- Tömb bejárása for ciklussal lehetséges.

Keresés

- a szekvenciális keresés bonyolultsága: $O(n)$
  - sima for ciklusos keresés
- a bináris keresés bonyolultsága: $\log_2(n)$
  - rendezett minta felezéses keresése

Rendezés

- a buborék rendezés bonyolultsága: $O(n^2)$
  - dupla for ciklusos rendezés $\binom{n}{2}$
- a quick sort bonyolultsága: $n \cdot \log_2(n)$
  - felezgetős keresés for ciklussal

Algoritmusok bonyolultsága

- P a polinomidőben megoldható problémákat reprezentálja
  - a gyakorlatban hatékonyan megoldható problémák osztálya
- NP a polinomidőben verifikálható problémák osztálya
  - a lehetséges megoldások polinomidőben előállíthatók nemdeterminisztikus Turing-géppel, majd determinisztikusan polinomidőben ellenőrizhető az, hogy az előállított lehetséges megoldás valóban megoldás-e
- megoldatlan probléma: $N \stackrel{?}{=} NP$
- utazó ügynök probléma
  - célja a legkisebb költségű út megtalálása városok között, minden várost pontosan egyszer érintve és visszatérve a kiindulási pontba
  - a városok közötti költség az Euklideszi távolságon alapul, a probléma szimmetrikus, a költségek konstansok
- $B(n)$ – bonyolultság

Rekurzió

- rekurzió esetén a függvényben, általában a visszatérésénél meghívjuk saját magát.
- rekurzívan megoldható problémák:
  - a faktoriális

```
int factorial(int n){
  if(n == 0) return 1;
  return n * factorial(n-1);
}

```

  - az aranymetszés

```
int golden_ratio(int n){
  if(n == 1 || n == 2) return 1;
  return golden_ratio(n-1) + golden_ratio(n-2)
}

```

  - Hanoi tornyai

```
void hanoi(int n, char from, char to, char tmp){
  if(n == 1){
    cout << "\nTedd at az 1. lemezt a(z) "
        << from << "rudrol a(z)" << to << " rudra.";
  }
  hanoi(n-1, from, tmp, to);
  cout << "\nTedd at a(z) " << n << " lemezt a(z) "
      << from << "rudrol a(z)" << to << " rudra.";
  hanoi(n-1, tmp, to, from);
}

```

9. Tétel – Az adatbázisok alapjai

Adatbázis

- adatok – nyers tények, feldolgozatlan információ
- információ – feldolgozott adat
  - információs rendszerek hozzák létre, keresik vissza, dolgozzák fel.
- az adatok és információk közötti különbség nem strukturális, hanem funkcionális
- az adatbázis adatok gyűjteménye, amelyet egy adatbázis-kezelő rendszer kezel, tehát nemcsak az adatok rendezett tárolását, hanem azok kezelését is lehetővé teszi, mert kapcsolatok nélkül az adatok eltérően értelmezhetőek
- az eltárolt adatok struktúrája a kezelő rendszer együttese
- az adatok között meghatározott kapcsolatok vannak
- alapfunkciók: létrehozás, adatok mentése, lekérdezések, adatvédelem
- a DBMS segítségével lehetséges a tárolt adatok definiálása, kezelése, karbantartása, felügyelete

Követelmények

- DDL – új adatok létrehozása adatdefiníciós nyelv segítségével
- SQL – meglévő adatok lekérdezése, módosítása lekérdező vagy adatmanipulációs nyelvvel
- támogassa az adatok hosszú távú, biztonságos tárolását
- felügyelje a több felhasználó által egy időben történő hozzáférését
- adatintegritás – érvényesség, helyesség, ellentmondás-mentesség
- rugalmasság – adatok egyszerűen módosíthatóak
- hatékonyság – gyors és hatékony keresés, módosíthatóság
- adatfüggetlenség – hardver és szoftverfüggetlenség
- adatbiztonság – védelem a hardver és szoftverhibák ellen
- adatvédelem – illetéktelen felhasználókkal szemben
- osztott hozzáférés – több felhasználó egyidejű hozzáférése
- integritási kényszerek – szabályok, amiket figyelembe kell venni
- tranzakciók – felhasználó általi változtatás nem végleges azonnal

Adatmodellezés

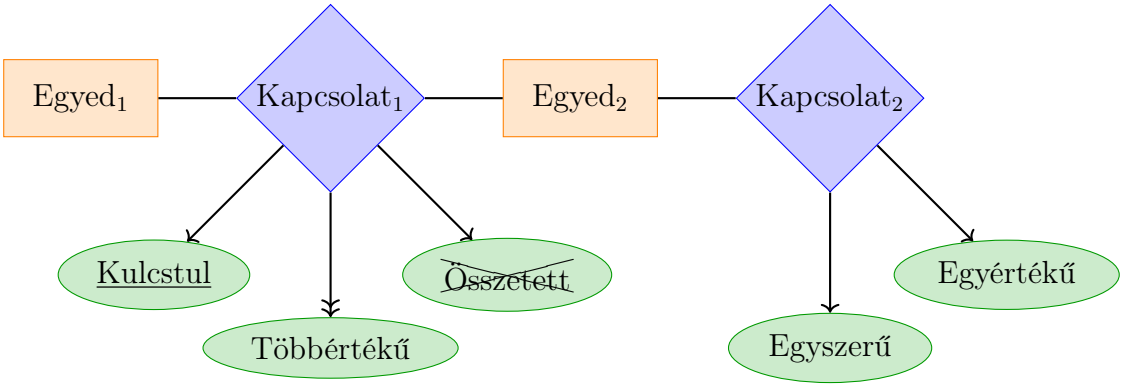
- az adatmodellezés segítséget nyújt a környezővilág megértésében és leképezésében, a lényeges jellemzők kiemelésében
- az adatmodell az adatok és az azok közötti összefüggések leírására szolgál
- a modell olyan mesterséges rendszer, amely felépítésében és viselkedésében megegyezik a vizsgált létező rendszerrel
- adatmodellnek nevezzük az adatok struktúrájának (felépítésének) leírására szolgáló modelleket.

Egyed-kapcsolat modell

- az ETK adatmodell a lenti három fogalom együttese
- egyed – az információs rendszert felépítő személyek, tárgyak, események
  - nem elszigetelten, hanem kapcsolatban állnak egymással és más objetumokkal
- tulajdonság – egyedeket jellemzik, egy érték, aminek tulajdonságtípusa van
  - azonosító – egyedi, nem ismétlődhet (ID)
  - leíró tulajdonság – ismétlődhet (név)
  - gyengén jellemző tulajdonság – lehet üres is (kedvenc szín)
  - kapcsoló tulajdonság – itt leíró, ott azonosító (szül hely)
- kapcsolat – 2 egyed közötti viszony
  - lehet többszintű, bonyolult, melyben több rendszer is kapcsolatban áll egymással

Kapcsolat típusok

- (1-1) – "egy az egyhez"
  - egy egyedtípus egy egyedéhez egy másik egyedtípus csak egyetlen egyede kapcsolódhat, és fordítva is igaz (osztály-osztályfőnök)
- (1-N) – "egy a többhöz"
  - egy egyedtípus egy egyedéhez egy másik egyedtípus több egyede is kapcsolódhat, de fordítva nem igaz (osztály-tanuló)
- (N-M) – "több a többhöz"
  - egy egyedtípus egy egyedéhez egy másik egyedtípus több egyede is kapcsolódhat, de fordítva is igaz (osztály-tanár)
- az adatbázis fogalma a kapcsolatok alapján:
  - véges számú egyedek és azok véges számú tulajdonságainak és kapcsolatainak adatmodell szerinti szervezett együttese



- a kapcsolat foka lehet...
  - Unáris – rekurzív
  - Bináris – két résztvevős
  - Tináris – három résztvevős

10. Tétel – Relációs adatbázismodell

Adatbázis szerkezetek

A leggyakoribb adatszerkezetek: hierarchikus, hálós, relációs.

Hierarchikus esetben csak egy a többhöz (1-N) kapcsolatok képezhetőek le. Az adatok a tárolt hierarchia szerint érhetőek el. Fastruktúrával szemléltethető, hiszen az adatokat alá-fölérendeltségi viszonnyal meghatározható szerkezettel írjuk le.

Hálós adatbázismodell esetén a kapcsolatok gráfokkal írhatóak le, ahol a csomópontokat élekkel kötjük össze. Csak a tárolt kapcsolat mentén járható be. A modellel egy a többhöz (1-N) és több a többhöz (N-M) kapcsolatot és leírhatunk.

Reláció jellemzői

Relációs modell esetén az adatokat táblázatos formában tároljuk. Nincsenek előre meghatározott kapcsolatok.

- egyszerűen értelmezhető, átlátható
- rugalmas, könnyen kezelhető
- relációk kezelése relációs algebrával

A reláció egy adattábla. (táblázat)

- rekord – a táblázat sorai
- attribútum – a táblázat oszlopai
- mező – a sorok és oszlopok metszetei

A reláció rekordjaiban tároljuk a logikailag összefüggő adatokat. A relációban tárolt rekordok száma a reláció számossága. Az oszlopokban azonos tulajdonságokra vonatkozó adatok jelennek meg. Egy tábla nem tartalmazhat azonos nevű oszlopot. Az oszlopok száma a reláció foka. Követelmények:

- minden rekordja különböző
- nincs két azonos attribútum
- minden rekord mezőszerkezete azonos
- a rekordok és attribútumok sorrendje tetszőleges

A relációs algebra műveletei

1. adatkezelő műveletek
   - adatbevitel, törlés, adatmódosítás
2. adatlekérdező műveletek
   - relációs algebra műveleteivel, mindig új relációt eredményez

A relációs algebra műveletei lehetnek egy vagy többoperandusúak. Az előbbit egy, az utóbbit pedig több reláción végezzük el.

- szelekció (kiválasztás)
  rekordokat választunk ki
- projekció (vetítés)
  attribútumokat választunk ki
- kiterjesztés
  matematikai műveletekkel új oszlop
- csoportosítás
  attribútumok alapján csoport, majd hozzá érték
- Descartes-szorzat
  2 reláció sorainak összes kombinációja
- összekapcsolás
  összekapcsolás attribútum alapján
- unió
- metszet
- különbség
  nem kommutatív

Az azonosítás történhet kulcs alapján, amely egyértelműen azonosítja az egyedet az egyedhalmazon belül. Amennyiben egyetlen attribútumból áll, akkor egyszerű, egyébként összetett. Megadható több kulcs is, amire szükségünk van az adott feladatnál, azt elsődleges kulcsnak nevezzük, a többi mind másodlagos. Idegen kulcs egy reláció olyan attribútumai, melyek egy másikban elsődlegesek.

SQL alapok

Hogy elkerüljük az anomáliákat, az adatbázisokat normalizálni szokták. Ennek lényege, hogy a változtatási anomáliák megszűnjenek. (módosítási, beírási, törlési) A normalizálásnak több szintje létezik. Minden relációs séma megköveteli legalább az első normálformát. Gyakorlatban a harmadikig szokták.

- 1. normál forma
  - ha a mezők függéseinek rendszerében létezik egy olyan kulcs, amelytől minden más mező függ, azaz minden mezője funkcionálisan függ a kulcsmező csoporttól
- 2. normál forma
  - nincs benne részleges függés, azaz bármely nem kulcs mező a teljes kulcstól függ, de nem függ a kulcs bármely részhalmazától
- 3. normál forma
  - nem áll fenn tranzitív függőség, azaz a nem kulcs mezők nem függnek egymástól, tehát nincs funkcionális függőség a nem elsődleges attribútumok között.

Definíció:

- `CREATE TABLE table_name (column1 datatype cond,...);`
  objektum létrehozása
- `DROP TABLE table_name;`
  objektum megszűntetése
- `ALTER TABLE table_name ADD|MODIFY (column1 datatype cond|cond);`
  objektum séma módosítása

PRIMARY KEY
NOT NULL
UNIQUE
CHECK cond
REFERENCING table_name

CHAR(n)
NUMBER(n,m)
DATE

Módosítás:

- `INSERT INTO table_name VALUES (field=value);`
  rekord felvitele
- `DELETE FROM table_name WHERE cond;`
  rekord törlése
- `UPDATE table_name SET field=value,... WHERE cond;`
  rekord módosítása

Adatok lekérdezése:

- `SELECT column1, column2, ... FROM table_name1, table_name2, WHERE cond;`

`GROUP BY`	csoportosítás
`HAVING cond`	megszorítás
`ORDER BY`	rendezés

- `SELECT column FROM table_name;` projekció
- `SELECT column FROM table_name WHERE cond;` szelekció
- `SELECT * FROM table_name1, table_name2;` Descartes-szorzat
- `SELECT expr column,... FROM table_name;` kiterjesztés
- `SELECT aggregation FROM table_name;` aggregáció megadása

`SUM(expr)`
`COUNT(expr)`
`MIN(expr)`
`AVG(expr)`
`MAX(expr)`

- `SELECT aggregation FROM table_name; GROUP BY expr;` aggregáció, csoportképzés
- `SELECT column FROM table_name; ORDER BY column1 mode1, ...;` eredmény rekordok rendezése

`ASC`
`DESC`

Adatok lekérdezése:

`=`	egyenlő
`<>,^=`	nem egyenlő
`>`	nagyobb
`>=`	nagyobb egyenlő
`<`	kisebb
`<=`	kisebb egyenlő

`BETWEEN x AND y`	adott értékek közé esik
`IN(a, b, c,...)`	értékek között van
`LIKE sample`	hasonlít a mintára

`LIKE 'a%'`	'a'-val kezdődik
`LIKE 'x_'`	'x'-val kezdődik, 2 betű
`LIKE '%a%'`	'a'-t tartalmaz
`LIKE '_a%x'`	2. betű 'a', 'x'-re végződik

`NOT`	tagadás
`AND`	és
`OR`	vagy

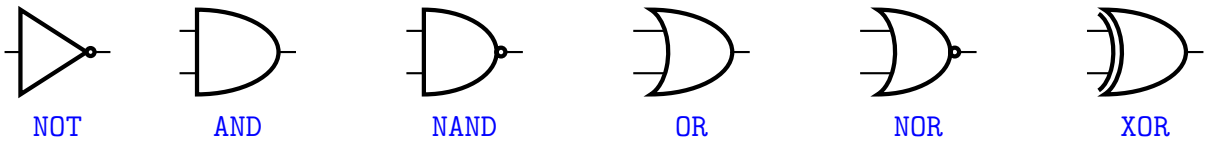
11. Tétel – A számítógép architektúrák alapjai

Boole függvények

- 2 változós Boole függvényekből 16 darab van
- n változósból 2^{2^n}
- AND, OR, NOT függvényekből az összes Boole függvény előállítható
- a NAND és a NOR önmagukban képesek az összeset előállítani

Logikai kapuk

A digitális áramkörök esetén az áramkör bármely pontján mérhető jeleknek csak 2 állapotát különbüztetjük meg. A digitális áramköröket logikai áramkörökkel modellezzük. Leírásukhoz Boole algebrát használunk. A logikai kapuk a logikai áramkörök építőkockái, logikai alapműveleteket valósítanak meg. Ezek kombinációjával további áramköröket tudunk felépíteni.



in	out
0	1
1	0

i_1	i_2	out
0	0	0
0	1	0
1	0	0
1	1	1

i_1	i_2	out
0	0	1
0	1	1
1	0	1
1	1	0

i_1	i_2	out
0	0	0
0	1	1
1	0	1
1	1	1

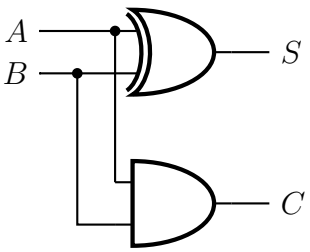
i_1	i_2	out
0	0	1
0	1	0
1	0	0
1	1	0

i_1	i_2	out
0	0	0
0	1	1
1	0	1
1	1	0

Kombinációs logikai hálózatok

Kombinációs logikai hálózatok esetén a kimeneti jelek értékei csak a bemeneti jelek pillanatnyi értékétől függenek. A kimenetek egy-egy függvénykapcsolattal írhatóak le.

Félösszeadó:



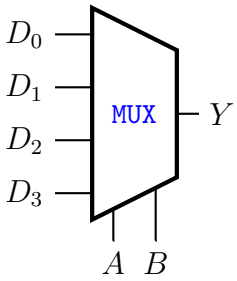
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- egy XOR és AND kapuval megvalósítható
- feladata 2 bit összeadása
- $S = \overline{A}B + A\overline{B}$ – összeg
- $C = AB$ – maradék (carry)

Teljes összeadó:

- feladata két bit és az előző helyi értékből származó maradék összeadása
- `bemenetek:` A, B, C_{in}, `kimenetek:` S, C_{out}
- $S = \overline{A}\overline{B}C_{\text{in}} + \overline{A}B\overline{C}_{\text{in}} + A\overline{B}\overline{C}_{\text{in}} + ABC_{\text{in}}$
- $C = \overline{A}BC_{\text{in}} + A\overline{B}C_{\text{in}} + AB\overline{C}_{\text{in}} + ABC_{\text{in}} = AB + BC_{\text{in}} + AC_{\text{in}}$

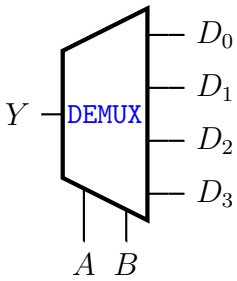
Multiplexer:



A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- feladata több bemenő jel közül az egyik kiválasztása
- 2^n db `bemenet`
  1 db `kimenet`
  n db `vezérlőbemenet`
- lehet még párhuzamos–soros adatkonverter

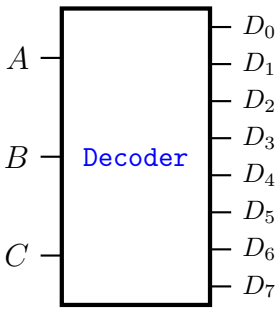
Demultiplexer:



A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- egy jel kapcsolása választható kimenetre
- 1 db `bemenet`
  2^n db `kimenet`
  n db `vezérlőbemenet`
- lehet még párhuzamos–soros adatkonverter

Címdekóder:



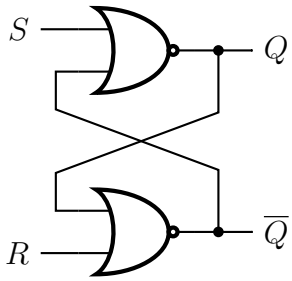
A	B	C	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

- feladata cím dekódolása
- `bemenet:` n bites szám
- `kimenet:` 2^n-ből választ ki 1-et

Szekvenciális logikai hálózatok

`Szekvenciális logikai hálózatok` esetén a kimenet nemcsak a bemeneti jelkombinációtól, hanem a hálózat állapotától is függ. (azaz a a hálózatra megelőzően ható jelkombinációktól) Léteznek `szinkron` (órajel) és `aszinkron` sorrendi hálózatok.

Flip-flop



S	R	Q
0	0	prev
0	1	0
1	0	1
1	1	?

- elemi sorrendi hálózatok
- két stabil állapotú billenő elemek
- állapotuk megegyezik a kimenettel
- regiszterek, `SRAM`, számlálók

12. Tétel – A számítógép architektúrák alapjai

A számítógép felépítése

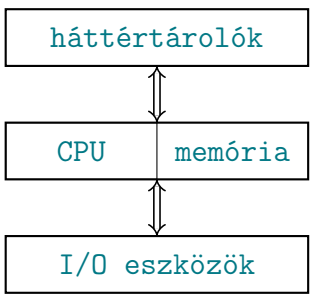
- Hardver – elektromos áramkörök, mechanikus berendezések, kábelek, csatlakozók, perifériák önmagában nem működőképesek
- Szoftver – számítógépet működőképessé tevő programok és azok dokumentációi
- Firmware – célprogram, mikrokóddal írt, készülék specifikus, hardverbe ágyazott szoftver, gyakran Flash ROM

A digitális számítógép olyan gép, amely a neki címzett utasítások alapján problémákat old meg. Az utasítássorozatot, amely leírja, hogy hogyan oldjunk meg egy feladatot, programnak nevezünk.

Gépi, nyelvi szintek:

0. digitális logikai szint
   - kapuk (gate)
1. mikroarchitektúra szintje
   - értelmezi a második szintet
   - ALU, regiszterek
2. gépi utasítás szintje (elektronikus áramkörök)
   - itt dől el a kompatibilitás kérdése
3. operációs rendszer gépi szintje
   - általában értelmezés
   - az utasításait az oprendszer, vagy közvetlen a 2. szint hajtja végre
4. assembly nyelvi szint (assembler)
   - szimbolikus leírás
5. probléma orientált nyelvi szint (fordító program)
   - ezek tényleges nyelvek (C, C++)

Neumann elvű számítógép felépítése:



- a CPU általános vezérlő, műveletvégző, adat-mozgató egység, végrehajtja a futó programok utasításait
- a memória a futó programok kódját, adatait tartalmazza
- a háttértárolók lehet mágneslemez, merevlemez, optikai tároló, szalagos tároló, félvezezős tároló (flash memória chip)
- a perifériák: monitor, billentyűzet, egér, nyomtató, kommunikációs vonalak, stb.

Számítógépek szokásos felépítése:

- a részegységek egy rendszersínen (rendszerbusz) keresztül kapcsolódnak egymáshoz
- tipikusan a renszerbusz, mikroprocesszor, memória, valamint az eszközvezérlők nagy része az alaplapon helyezkedik el
- bővítőkártyák is tartalmazhatnak eszközvezérlőket
- az eszközvezérlő képes lehet DMA-t végezni; ha kész, megszakítást vált ki
- 3 típusú információ áramolhat: cím, adat, vezérlő

Buszok:

- a buszok jellemzésére az adat- és címvonalak számát, az adatátvitel jellemzőit, időzítés adatait, a vezérlőjelek típusait, funkcióit kell megadni
- a cím lehet memóriacím, vagy IO eszköz címezhető
- a vezérlőjelek lehetnek...
  - adatátvitelt vezérlő jelek
    - cím a sínen – memória, periféria (M/IO)
    - adat a sínen – írás, olvasás (R/W)
    - átvitel vége – szó, byte átvitel (WD/B)
  - megszakítást vezérlő jelek
  - sínvezérlő jelek (kérés, foglalás, visszaigazolás)
  - egyéb (órajel, ütemezés, táp)

Memóriák

Memória hierarchia:

- regiszter
  gyorsítótár
  központi memória
  mágneslemez
  szalag, optikai lemez

Csoportosítás:

1. információ elérése alapjám
   - cím szerinti hozzáférés
   - tartalom szerinti hozzáférés (cache)
2. hozzáférés belső szervezés alapján
   - szekvenciális memóriák
   - tetszőleges sorrendben címezhető memóriák
     - csak olvasható memóriák (ROM)
     - írható-olvasható memóriák (RAM)

Tetszőleges sorrendben címezhető memóriák:

- sor és oszlopdekóderek
- író, olvasó
- memóriacella egy bit tárolására képes

ROM típusok:

Minden egyes típus egyedi karakterisztikával bír, de két dologban közösek. Az eltárolt adatok ezekben a lapkákban nem illékonyak, azaz nem vesznek el, amikor kikapcsoljuk az áramot. Az eltárolt adatok megváltoztathatatlanok, vagy speciális műveletet igényel a változtatás. (Ellentétben a RAM-mal, melynél könnyű a változtatás)

- ROM – Read Only Memory
  - a gyártó programozza
- PROM – Programmable Read Only Memory
  - felhasználó egyszer programozhatja, azaz megfelelő készülékkel kiégetheti a cellákban lévő tranzisztorok bekötő vezetékeit
- EPROM – Erasable Programmable Read Only Memory
  - UV fénnyel törölhető, majd külön készülékkel újra írható a tartalma
  - régebben a ROM BIOS ilyen memóriában helyezkedett el
- EEPROM – Electrically Erasable Programmable Read Only Memory
  - elektromosan törölhető,majd külön készülékkel újra írható a tartalma
- FLASH – Flash/Villanó Memória
  - Olyan EEPROM, melyet számítógép is képes törölni, majd újreírni.
  - Pendrive-okban, fényképezőgépekben

RAM típusok:

- SRAM – Static Random Access Memory
  - a tápfeszültség biztosításával korlátlan ideig megőrzi az információt
  - a memóriacellában egy flip-flop található
  - kisebb integráltságú (nagyobb méretű egy cella, mint a DRAM esetén)
  - nagyon gyors: cache
- DRAM – Dynamic Random Access Memory
  - az információt egy pici kondenzátor tárolja
  - a szivárgás miatt rövid időn belül elveszítené a töltését, ezért időközönként (néhány ms) frissíteni kell a tartalmát
  - nagy integráltságú, a PC-k memóriája ilyen

CPU részei

A CPU (Central Processing Unit – központi feldolgozó egység) a memóriából olvassa a végrehajtás alatt lévő program bináris utasításait. Az utasításkészlete fontos jellemzője. A mikroprocesszor egy chipen kialakított áramkör, mely a számítógép CPU-jának a funkcióját látja el. Részei:

- ALU – aritmetikai és logikai műveletek végzése
  - összeadás, kivonás, fixpontos szorzás, osztás (léptetések), lebegőpontos aritmetikai műveletek (korábban koprocesszor), egyszerű logikai műveletek
- utasítás dekódoló és vezérlő egység
  - Felismeri, elemzi (dekódolja) a gépi nyelvű program utasításait, az utasítások alapján működteti a CPU többi egységét, illetve képezi a szükséges címeket.
- regiszterek – chipen belüli, közvetlen elérésű tároló elemek
  - feladatuk műveletvégzéskor az operandusok tárolása, illetve a címek előállítása

8086 processzor:

- szegmensregiszterek
  - CS – Code Segment – kódszegmens regiszter
  - SS – Stack Segment – veremszegmens regiszter
  - DS – Data Segment – adatszegmens regiszter
  - ES – Extra Segment – extra adatszegmens regiszter
- vezérlő regiszterek
  - IP – Instruction Pointer – utasítás mutató
  - SP – Stack Pointer – verem mutató
  - BP – Base Pointer – bázis mutató
  - SI – Source Index – forrás index
  - DI – Destination Index – cél index
- általános célú regiszterek - adatregiszterek
  - AX – akkumulátor regiszter (AH, AL)
  - BX – bázis regiszter (BH, BL)
  - CX – számláló regiszter (CH, CL)
  - DX – adatregiszter (DH, DL)
  - műveletvégzéskor az operandusok tárolására
- flag-ek – jelzőbitek, melyek...
  - vagy a legutóbb elvégzett aritmetikai műveletek eredményétől függően vesznek fel értékeket, vagy az processzor állapotára utalnak
  - a feltételes ugró utasítások a flag-eket használják feltételre

  - aritmetikai flag-ek: előjel flag (sign), zéró flag (zero), paritás flag (parity), átvitel flag (carry) (legmagasabb helyiértéken képződött maradék)
  - processzor állapotára utalóak: trap flag (program utasításonkénti végrehajtása), interrupt flag (megszakítás, a hardver egységek felől érkező megszakítások eljutnake a processzorhoz), overflow flag (túlcsordulás),

- CPU címzése:
  - memóriacímek: a program utasításainak beolvasására, adatainak írására, olvasására
  - IO címek: a perifériákkal való kommunikációra

- utasításkészlet
  - mikroprocesszorok egyik legfontosabb jellemzője, hogy milyen utasításokat ismernek, milyen a gépi nyelvük
  - a gépi utasítások bináris jelsorozatok
  - az assembly nyelv a gépi utasításokat mnemonikkal helyettesíti

- assembly utasítások

MOV	adatmozgatás	JMP flag	ugrás
ADD	összeadás	JZ flag	ugrás[2]
SUB	kivonás	CMP	összehasonlítás
MUL	szorzás[1]	PUSH, POP	verem
DIV	osztás[1]	LDA, STA	akkumulátor[3]

1. MUL → AL·XX=AX DIV → AX/XX=AL
2. JZ → akkor ugrik ha a Zero flag aktív
3. LDA → 2 byte-ot másol a memóriából az akkumulátorba (LoaD Accumulator)
   STA → az akkumulátor tartalmát a memóriába másolja (STore Accumulator)

Utasítás ciklus

1. fetch (elérés)
   - utasítás kód beolvasása
   - utasítás kód értelmezése (dekódolás)
   - operandusok beolvasása
2. execute (végrehajtás)
   - műveletvégzés (ALU)
   - eredmény tárolása
   - következő utasítás címének kiszámítása

Szubrutinhívás

Szubrutinhívás esetén a program máshol folytatódik. (alprogramra ugrunk) CALL, RET utasításpár. Ahhoz, hogy vissza tudjunk térni a megfelelő helyre, el kell menteni a PC (utasításszámláló) értékét a stack-be (PUSH)

Interrupt

A megszakítás (interrupt) egy erőltetett vezérlésátadás, ugrás egy megszakítást kezelő rutinra. Előidézheti egy mikroprocesszorban előforduló esemény. (zéróosztás) Érkezhet megszakítás egy hardver egység felől is. (adatok beolvasása a memóriába megtörtént) Program is tartalmazhat megszakítási utasítást. (oprendszeri szolgáltatás) Fő okai:

- processzor megszakítás
- hardver megszakítás (IRQ: interrupt request) (lehet maszkolható, vagy nem maszkolható)
- szoftveres megszakítás

A szubrutinhíváshoz hasonlóan az utasításszámláló értékét a stack-be mentjük.

Közvetlen memória hozzáférés

A DMA (Direct Memory Access) a memória és egy periféria (merevlemez) közötti közvetlen adatátvitel. A DMA vezérlő irányítja az adatforgalmat, így a CPU közben egy másik program kódját futtatja. DMA nélkül az adatokat a processzoran kellene átvezetni, amely nagyon időigényes lenne. Kezdetben szükség van az iniciálásra, de utána a folyamat a processzor igénybevétele nélkül folytatódik.

13. Tétel – Az operációs rendszerek alapjai

Az operációs rendszer céljai, feladatai

Az operációs rendszer egy programrendszer, mely közvetítő szerepet lát el a számítógép felhasználója és a számítógép hardvere között.

Céljai:

- felhasználói programok végrehajtása, felhasználói feladatmegoldás megkönnyítése
- a számítógép rendszer használatának megkönnyítése
- a számítógép hardver kihasználásának hatékonyabbá tétele

Számítógép rendszerek komponensei:

- hardver: az alapvető számítási erőforrásokat nyújtja
- operációs rendszer: koordinálja és vezérli a hardver erőforrások különböző felhasználók által történő használatát
- alkalmazói programok: definiálja azt a módot, ahogyan az egyes rendszer-erőforrásokat a felhasználók számítási problémáinak megoldásához fel kell használni. (fordítók, adatbázis kezelők, videó játékok, ügyviteli programok)
- felhasználók: emberek, gépek, más számítógépek

Komponensei:

- folyamat kezelés
- másolagos tár kezelés
- fájl kezelés
- hálózat-elérés támogatása
- memória gazdálkodás
- IO rendszer kezelés
- védelmi rendszer
- parancs-interpreter rendszer

Folyamatok kommunikációja

A folyamat (process) egy végrehajtás alatt lévő program. Bizonyos erőforrásra van szüksége, hogy feladatát megoldhassa. (CPU, memória, állományok, IO berendezések) Az operációs rendszer az alábbi tevékenységekért felel:

- folyamat létrehozása/törlése
- folyamat felfüggesztése/újraindítása
- eszközök biztosítása (folyamatok szinkronizációjához, kommunikációjához)

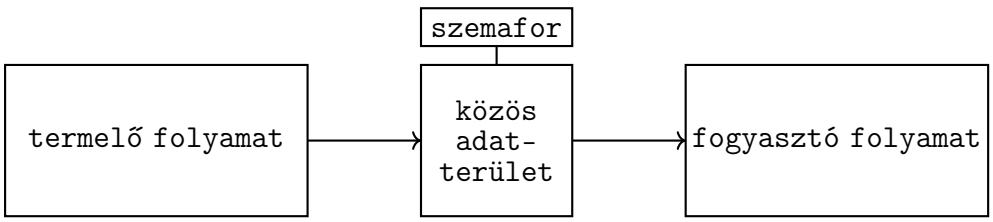
A folyamat a multiprogramozott operációs rendszerek alapfogalma. A folyamaton általában műveletek meghatározott sorrendjét értjük. A folyamat elkezdődik és be is fejeződik. Minden részművelet végrehajtása csak akkor kezdődhet meg, ha az előző részművelet végrehajtása már befejeződött.

- független folyamat – egymás működését semmilyen módon nem befolyásolják
- versengő folyamat – nem ismerik egymást, de közös erőforráson kell osztozniuk
- együttműködő folyamat – ismerik egymást, információt cserélnek, együtt dolgoznak

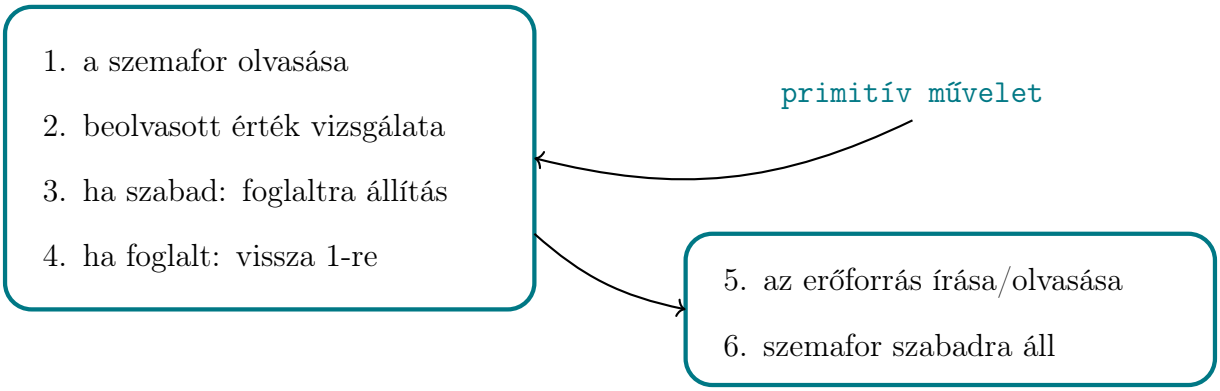
Több egymással párhuzamosan futó folyamat gyakran kommunikál közösen használt memóriaterületek segítségével. Ezek a területek nem érhetőek el egyidejűleg a folyamatok számára. Az egyidejű hozzáférés kizásása szemaforok segítségével történik.

Termelő-fogyasztó probléma

Legyen egy termelő és egy fogyasztó folyamatunk, melyek közös adatterületet használnak. Ilyenkor fellép a kölcsönös kizárás igénye, hiszen adott memóriaterületet egyszerre csak egy process használhat. Ilyenkor a vezérlés szemafor segítségével történik.



Mielőtt a folyamat használni kezdené a közös erőforrást, ellenőriznie kell, hogy szabad-e. Csak akkor kezdheti el használni, ha a szemafor szabadot jelzett. A primitív megszakíthatatlan, oszthatatlan művelet. Legyen P (foglalttá állítás) és V (szabaddá állítás) primitívek **S** bináris szemafor.

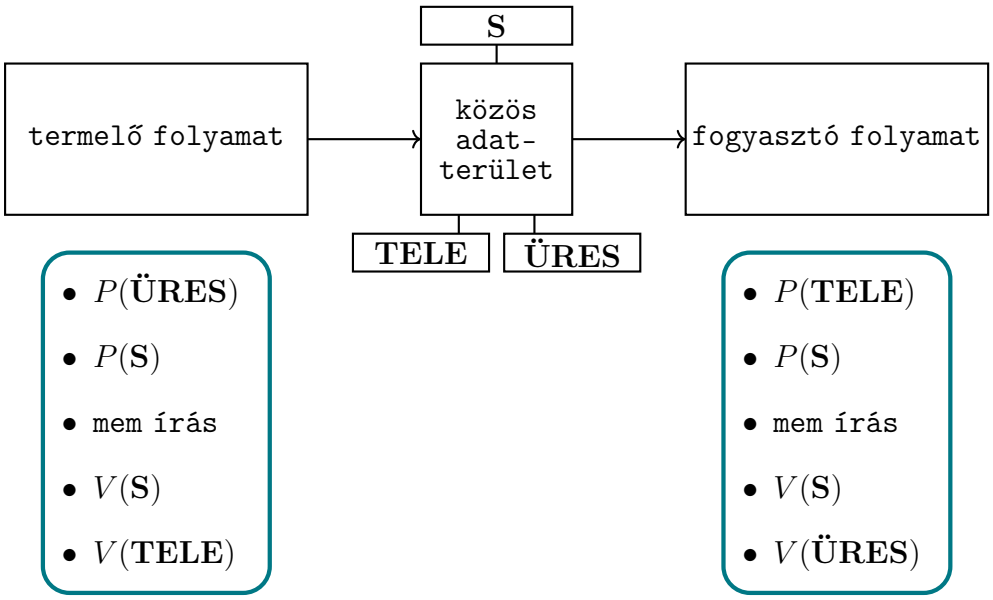


$P(\mathbf{S}) \rightarrow$ `memória írás/olvasás` $\rightarrow V(\mathbf{S})$

Postaláda kezelés

A postaláda olyan közös adatterület, ahová egynél több üzenet írható. Kezelése hasonlít az előző problémához. A vezérléséhez 3 szemafor szükséges, P és V primitívek:

- **S** – a kölcsönös kizárást valósítja meg, bináris (`0-foglalt`, `1-szabad`)
- **TELE** – a tele helyek száma, nem bináris (értéke: `0...N`, kezdetben `0`)
- **ÜRES** – az üres helyek száma, nem bináris (értéke: `0...N`, kezdetben `N`)
- P – szemafor értékét eggyel csökkenti (`foglalttá állítás`)
- V – szemafor értékét eggyel növeli (`szabaddá állítás`)



Szemaforok

A közös adatterületet egyszerre csak egy folyamat használhatja. (kölcsönös kizárás) A vezérlés szemaforok segítségével történik. Lehetnek binárisak és nem binárisak.

14. Tétel – Az operációs rendszerek alapjai

Holtpont

A holtpont egy rendszernek egy olyan állapota, ahonnan külső beavatkozás nélkül nem tud elmozdulni. Holtpont akkor fordulhat elő, amikor a folyamatok egy adott halmazában minden egyes elem leköt néhány erőforrást, és ugyanakkor várakozik is másokra.

Holtpont kezelése

- strucc algoritmus – nem teszünk semmit
- detektálás és feloldás – észrevesszük, ha holtpont alakult ki, és megpróbáljuk feloldani
- megelőzés – strukturálisan holtpontmentes rendszert tervezünk

Holtpont észlelése

Tegyük fel, hogy 4 folyamatunk és 10 egység erőforrásunk van. A helyzet a következő:

	foglal	kér
P1	4	4
P2	1	0
P3	3	4
P4	1	2

Radikális lépés, ha az összes folyamatot felszámoljuk. Kiméletes, ha megnézzük, van-e menthető folyamat, esetleg prioritás, vagy éppen mekkora része lett már az adott folyamat feladatának elvűgezve. Minden esetben biztosítani kell a folyamatok visszaállíthatóságát.

Holtpont megelőzés

Biztonságosnak nevezzük azokat a folyamat-erőforrás rendszereket, amelyekben létezik a folyamatoknak (legalább egy) olyan sorrendje, amely szerint végrehajtva őket, azok maximális erőforrás igénye is kielégíthető. Biztonságos állapotban nem lehetséges holtponti állapot kialakulása. A biztonságos állapotot bankár algoritmussal ellenőrizhetjük. (bankban is ilyen módszert alkalmaznak)

Bankár algoritmus

1. adatok mátrixos felírása
2. az igények és a szabad erőforrások kiszámítása
3. megnézzük, hogy valamelyik folyamat kielégíthető-e
4. újraszámítás, folytatás...

Ha találunk egy olyan sorrendet, amelyben a folyamatok erőforrás igénye kielégíthető, akkor a rendszer biztonságos állapotban van.

15. Tétel – Az operációs rendszerek alapjai

Ütemezési algoritmusok

Többfeladatos (multitask) rendszereknél a folyamatok közötti átkapcsolást, azaz a környezetváltást az alacsony szintű ütemezési algoritmusok végzik. Általában a gyakorlatban többféle módszer kombinációját alkalmazzák. Alap algoritmusok:

- FCFS – First Come First Served
- SJF – Shortest Job First
- RR – Round Robin

	érk. idő	CPU igény
P1	0	14
P2	7	8
P4	20	10
P3	11	36

Előbb jött – előbb fut algoritmus

Az FCFS algoritmus esetén a folyamatok érkezési sorrendjükben kapják meg a processzort. Előnye, hogy ez a legegyszerűbb. Hátránya, hogy a várakozási idő nagymértékben függ az érkezési időtől. (csorda hatás, lassú kamion effektus)

	érk. idő	CPU igény	kezd. időpont	bef. időpont	várakozási idő
P1	0	14	0	14	0
P2	7	8	14	22	7
P3	11	36	22	58	11
P4	20	10	58	68	38

Jelen esetben az átlagos várakozási idő: $0 + 7 + 11 + 38 = 56 \quad \rightarrow \quad 56/4 = 14$.

A legrövidebb előnyben algoritmus

Az SJF algoritmus esetén a CPU-t egy folyamat befejezése után a legrövidebbnek adja oda. Előnye, hogy így a legrövidebb a várakozási idő. Hátránya, hogy tudni kell előre a folyamatok hosszát, illetve hogy kiéheztei a hosszú folyamatokat.

	érk. idő	CPU igény	kezd. időpont	bef. időpont	várakozási idő
P1	0	14	0	14	0
P2	7	8	14	22	7
P4	20	10	22	32	2
P3	11	36	32	68	21

Jelen esetben az átlagos várakozási idő: $0 + 7 + 2 + 21 = 30 \quad \rightarrow \quad 30/4 = 7,5$.

Körbenforgó algoritmus

Az RR algoritmus esetén a folyamatokat egy zárt körbe szervezzük, és minden folyamat egy előre rögzített maximális időre kapja meg a processzort, majd visszaáll a sor végére. Kombinálható prioritások bevezetésével. (minden prioritási szintnek "saját köre" van) Előnye, hogy egyszerű, és nincsen kiéheztetés. Hátránya, hogy az időszeletek lejártakor a folyamat állapotát el kell menteni. (időveszteség)

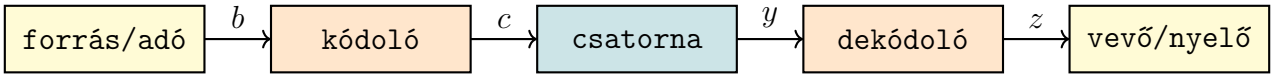
	érk. idő	CPU igény	kezd. időpont	bef. időpont	várakozási idő
P1	0	14	0	10	0
P2	7	8	10	18	3
(P1)	10	4	18	22	8
P3	11	36	22	32	11
P4	20	10	32	42	12
(P3)	32	26	42	52	10
(P3)	42	16	52	62	0
(P3)	52	6	62	68	0

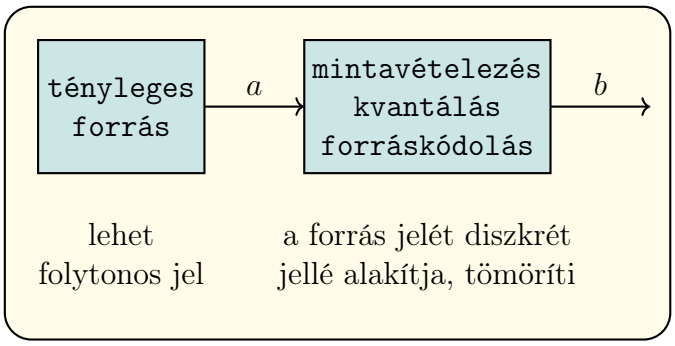
Jelen esetben az átlagos várakozási idő: $44/4 = 11$. (Az időszelet: 10)

16. Tétel – Az információelmélet alapjai

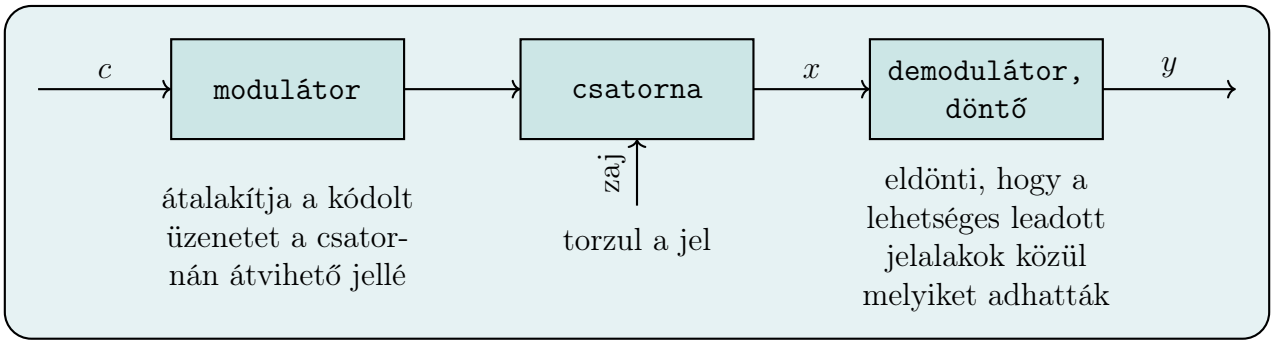
Shannon hírközlési modellje

A hírközlés során egy üzenetet juttatunk el egy tér- és időbeli pontból egy másikba.

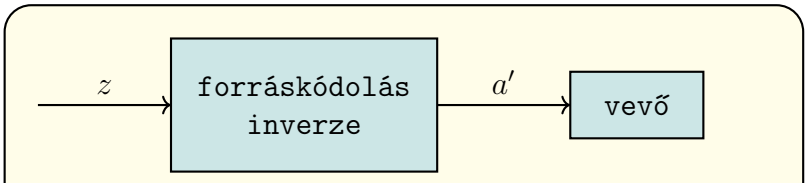




a `csatornakódolás` (hibajavító) lehetővé teszi a zajos csatornán való biztonságos(abb) üzenetátvitelt, a keletkező hibák jelzését és javítását



a `dekódoló` kijavítja, és / vagy jelzi a vett jelek hibáit. Elvégzi a `csatornakódolás` inverz műveletét.



Az információ

Az információ valamely véges számú, előre ismert esemény közül annak a megnevezése, hogy melyik következett be. Értéke azonos azzal a bizonytalansággal, melyet megszűntet.

Hartley:

m számú, azonos valószínűségű esemény közül egy megnevezésével nyert információ.

$$I = \log_2(m)$$

Shannon:

Shannon szerint minél váratlanabb az esemény, bekövetkezése annál több információt jelent. Legyen $A = \{A_1, A_2, \ldots, A_m\}$, A_i esemény valószínűsége p_i. Az A_i esemény megnevezésével nyert információ ekkor:

$$I\left(A_i\right) = -\log_2\left(p_i\right)$$

Az entrópia

Az entrópia az információ várható értéke.

$$H(p_1, p_2, \ldots, p_m) = \langle I(A) \rangle = \sum_{i=1}^{m} p_i \cdot I\left(A_i\right) = -\sum_{i=1}^{m} p_i \cdot \log_2(p_i)$$

Az entrópia tulajdonképpen annak a kijelentésnek az információtartalma, hogy az m db egymást kizáró esemény közül az egyik bekövetkezett.

Forráskódok

- a forrás kimenetén véges sok elemből álló $A = \{A_1, A_2, \ldots, A_m\}$ halmaz elemei jelenhetnek meg
- forrásábcé – maga az A halmaz
- üzenet – az A elemeiből képzett véges $A^{(1)}\ A^{(2)}\ \ldots\ A^{(m)}$ sorozatok
- $\mathcal{A}$ – a lehetséges üzenetek halmaza
- a kódolt üzenetek egy $B = \{B_1, B_2, \ldots, B_s\}$ szintén véges halmaz elemeiből épülnek fel
- kódábécé – maga a B halmaz
- kódszavak – a B elemeiből képzett véges $B^{(1)}\ B^{(2)}\ \ldots\ B^{(s)}$ sorozatok
- $\mathcal{B}$ – a lehetséges kódszavak halmaza
- az $f : A \to \mathcal{B}$ illetve $F : \mathcal{A} \to \mathcal{B}$ függvényeket forráskódoknak nevezzük. Az f leképezés a forrás egy-egy szimbólumához rendel egy-egy szót.

Egyértelműen dekódolható kódok

Egy f forráskód egyértelműen dekódolható, ha minden egyes B-beli sorozatot csak egyféle A-beli sorozatból állít elő, azaz a neki megfelelő F invertálható. Nem elég, hogy f invertálható legyen.

- ! $A = \{a, b, c\}$, $B = \{0, 1\}$ és $f(a) = 0$, $f(b) = 1$, $f(c) = 01$. Ekkor f invertálható, de a 01 kódszót dekódolhatjuk $f(a)f(b) = 01$ ezerint ab-nek, vagy $f(c) = 01$ szerint c-nek is.

Az állandó kódszóhosszú kódok egyértelműen dekódolhatóak, megfejthetőek, de nem elég gazdaságosak.

Prefix kód

Az f kód prefix, ha a lehetséges kódszavak közül egyik sem folytatása a másiknak, vagyis bármely kódszó végéből bármekkora szegmenst levágva nem kapunk egy másik kódszót. Prefix kód egyértelműen dekódolható.

- ! $A = \{a, b, c\}$, $B = \{0, 1\}$ és $f(a) = 0$, $f(b) = 10$, $f(c) = 110$. Ha az *abccab* üzenetet kódoljuk, akkor a 010110110010 kódsorozatot kapjuk. A kódból az üzenet visszafejtése nagyon egyszerű a prefix tulajdonság miatt.
- ! $A = \{a, b, c, d\}$, $B = \{0, 1\}$ és $f(a) = 0$, $f(b) = 01$, $f(c) = 011$, $f(d) = 0111$. Ez a kód nem prefix, de egyértelműen dekódolható, hiszem a 0 karakter egy új kódszó kezdetét jelzi.

17. Tétel – Az információelmélet alapjai

Forráskódolás

Kódszavak átlagos szóhossza:

Az olyan $f : A \to \mathcal{B}$ kódokat, melyek különböző A-beli szimbólumokhoz más-más hosszúságú kódszavakat rendelnek, változó szóhosszúságú kódoknak nevezzük. $f(A_i) = B^{(1)}\ B^{(2)}\ \dots\ B^{(\ell_i)}$ $\mathcal{B}$-beli sorozat (kódszó) hossza: ℓ_i. Egy f kód átlagos szóhossza ℓ_i várható értéke:

$$L(A) = \sum_{i=1}^{n} p(A_i) \cdot \ell_i = \sum_{i=1}^{n} p_i \cdot \ell_i$$

Shannon forráskódolási tétele:

Minden $A = \{A_1, A_2, \dots, A_n\}$ véges forrásábcéjű forráshoz található olyan s elemű kódábécével rendelkező $f : A \to \mathcal{B}$ kód, amely az egyes forrásszimbólumokhoz rendre $\ell_1, \ell_2, \dots, \ell_n$ szóhosszúságú szavakat rendel, és ...

$$\frac{H(A)}{\log_2(s)} \leq L(A) < \frac{H(A)}{\log_2(s)} + 1$$

Az olyan kódok, melyekre ez teljesül, azok optimális kódok.

Forráskódolás: A jól tömöríthető eljárásokra igaz, hogy ha $p_i \geq p_j$, akkor $\ell_i \leq \ell_j$. Ha az f bináris kód prefix, akkor ...

- a leggyakoribb forrásábécébeli elemhez fog a legrövidebb kódszó tartozni
- a második leggyakoribbhoz eggyel hosszabb kódszó
- ...
- a két legritkábban előforduló betűhöz pedig azonosan hosszú kódszó fog tartozni, és csak az utolsó karakterben fog e két szó különbözni

Huffman-kód

A Huffman-kód a legrövidebb átlagos szóhosszú bináris prefix kód.

1. valószínűségek szerint sorba rendez
2. a két legkisebb valószínűséhű szimbólumot összevonja. Az összevont szimbólum valószínűsége az eredeti két valószínűség összege.
3. az első 2 lépés ismételgetése
4. a kapott gráf minden csomópontja előtti két élt megcímkézi 0-val és 1-gyel
5. a kódfa gyökerétől elindulba megkeresi az adott szimbólumhoz tartozó útvonalat, kiolvassa az éleknek megfelelő biteket. A kapott bitsorozatot rendeli a szimbólumhoz kódszóként.

Csatornakódolás

$\mathbb{C}^n$ vektortér, $\boldsymbol{c} \in \mathbb{C}^n$ és $\boldsymbol{v} \in \mathbb{C}^n$ vektorok. A csatorna a rá bocsájtott $\boldsymbol{c} = c^{(1)}, c^{(2)}, \ldots, c^{(n)}$ szimbólumsorozatból egy $\boldsymbol{v} = v^{(1)}, v^{(2)}, \ldots, v^{(n)}$ szimbólumsorozatot csinál.

Hamming-távolság

A Hamming-távolság $\boldsymbol{c}$ és $\boldsymbol{v}$ eltérésének mérésére definiált távolság. Alatta azon i pozíciók számát értjük, ahol $c^{(i)} \neq v^{(i)}$. Jele: $\mathrm{d}(\boldsymbol{c}, \boldsymbol{v})$. Teljesülnek az alábbiak:

$$\mathrm{d}(\boldsymbol{c}, \boldsymbol{v}) \geq 0, \qquad \mathrm{d}(\boldsymbol{c}, \boldsymbol{c}) = 0 \qquad \mathrm{d}(\boldsymbol{c}, \boldsymbol{v}) = \mathrm{d}(\boldsymbol{v}, \boldsymbol{c}) \qquad \mathrm{d}(\boldsymbol{c}, \boldsymbol{v}) \leq \mathrm{d}(\boldsymbol{c}, \boldsymbol{w}) + \mathrm{d}(\boldsymbol{w}, \boldsymbol{v})$$

Hibajelzés

Az egyszerű hibázás esetén nem tudjuk, hogy melyik pozíciókban rontott a csatorna, csak azt, hogy hány darab hiba van. törléses hiba esetén ismerjük a hibázások helyét is, csak azt nem, hogy mennyire romlott el azokon a helyeken a jel.

Legyen K a lehetséges kódszavak halmaza. Ekkor egy K kód kódtávolsága a kódszavak közötti Hamming-távolság minimuma.

$$d_{\min} = \min_{\boldsymbol{c} \neq \boldsymbol{c'}; \boldsymbol{c}, \boldsymbol{c'} \in K} \{\mathrm{d}(\boldsymbol{c}, \boldsymbol{c'})\}$$

Hibajelzés lehetséges, ha a $\boldsymbol{c}$ kódszavunkból keletkezett $\boldsymbol{v}$ nem egy másik érvényes kódszó. Ha ν a hibák száma, akkor $\nu < d_{\min}$ hibát lehet mindenképp jelezni. Hibajelzés után általában megismétlik az üzenetet.

Törléses hiba javítása:

Ebben az esetben tudjuk a hibák helyét. A $\boldsymbol{v}$ hibásan vett vektort abba a kódszóba javítjuk, amelyik a hibás pozícióktól eltekintve azonos $\boldsymbol{v}$-vel. Ha több ilyen van, nem tudunk javítani. Ha a két legközelebbi kódszóból $d_{\min}$ komponenst a megfelelő helyről törlünk, akkor azonos maradékot kapunk, ennél kevesebb elem törlésével sehogy sem kaphatunk azonos maradékot. Így $\nu \leq d_{\min} - 1$ törléses hiba javítható.

Törléses hiba javítása:

Ebben az esetben nem tudjuk a hibák helyét. A $\boldsymbol{v}$ hibás vett vektort abba a $\boldsymbol{c}$ szóba javítjuk amelyre $\mathrm{d}\{\boldsymbol{c}, \boldsymbol{v}\}$ a legkisebb. Ha több ilyen van, akkor nem tudunk javítani. A javítóság feltétele:

$$\mathrm{d}(\boldsymbol{c}, \boldsymbol{v}) < \mathrm{d}(\boldsymbol{c'}, \boldsymbol{v}) \qquad \rightarrow \qquad \nu \leq \frac{d_{\min} - 1}{2}$$

Valszám alapok

- ellentett esemény – kísérlet minden A-n kívüli eseménye
- valószínűség – $p(A)$
- szorzat – együttes bekövetkezési valószínűség
- összeg – vagylagos bekövetkezési valószínűség
- ha A és B függetlenek – $p(A \cdot B) = p(A) \cdot p(B)$
- ha A és B függők
  - $p(A \cdot B) \leq p(A) \cdot p(B)$
  - $p(A + B) = p(A) + p(B) - P(A \cdot B)$
- feltételes valószínűség
  - $p(A|B) = \dfrac{P(A \cdot B)}{p(B)}$ - Ha B jelet vettünk, akkor annak a valószínűsége, hogy A jel volt a csatorna bemenetén
  - $p(B|A) = \dfrac{P(A \cdot B)}{p(A)}$ - ha A jelet adok, milyen B kerül a csatorna kimenetére
- várható érték – $\langle A \rangle = \sum_{i=1}^{m} p_i \cdot A_i$
- szórás – $D(A) = \sqrt{\langle (A - \langle A \rangle)^2 \rangle}$
- korreláció – $R(A, B) = \dfrac{\langle (A - \langle A \rangle) \cdot (B - \langle B \rangle) \rangle}{D(A) \cdot D(B)}$