RÃ©szproblÃ©mÃ¡ra bontÃ³ algoritmusok (mohÃ³, oszd-meg-Ã©s-uralkodj, dinamikus programozÃ¡s), rendezÅ algoritmusok, grÃ¡falgoritmusok (szÃ©lessÃ©gi- Ã©s mÃ©lysÃ©gi keresÃ©s, minimÃ¡lis feszÃtÅfÃ¡k, legrÃ¶videbb utak)

RÃ©szproblÃ©mÃ¡ra bontÃ³ algoritmusok

Oszd meg Ã©s uralkodj

- a feladatot tÃ¶bb rÃ©szfeladatra osztjuk
- a rÃ©szfeladatok hasonlÃ³ak az eredeti feladathoz, de kisebbek
- rekurzÃvan megoldjuk a kisebb rÃ©szfeladatokat
- a megoldÃ¡sokat Ã¶sszevonjuk, Ã©s az adja a vÃ©gsÅ megoldÃ¡st

FelosztÃ¡s: hogyan osztjuk tÃ¶bb rÃ©szfeladatra

UralkodÃ¡s: a rÃ©szfeladatokat rekurzÃv mÃ³don megoldjuk

ÃsszevonÃ¡s: a rÃ©szfeladatok megoldÃ¡sait Ã¶sszevonjuk

PÃ©lda

FelezÅ-csÃºcskeresÅ algoritmus

Input: egy szÃ¡msorozat Output: van-e benne csÃºcs?

Algoritmus: az n mÃ©retÅ± sorozat helyett vizsgÃ¡ljunk egy $(n-1)/2$ mÃ©retÅ±t, Ã©s ebben keressÃ¼nk csÃºcsot, majd ezt folytatjuk rekurzÃvan

Dinamikus programozÃ¡s

PÃ©nzvÃ¡ltÃ¡si feladat

Input: $P_1, P_2, \dots, P_n$ tÃpusÃº pÃ©nzÃ©rmÃ©k, F forint Output: legkevesebb hÃ¡ny Ã©rmÃ©vel fizethetÅ ki pontosan F forint?

Dinamikus programozÃ¡s akkor, ha a rÃ©szproblÃ©mÃ¡k nem fÃ¼ggetlenek, hanem vannak kÃ¶zÃ¶s rÃ©szeik

Alapgondolat: a mÃ¡r megoldott rÃ©szproblÃ©mÃ¡k optimÃ¡lis megoldÃ¡sait megjegyezzÃ¼k

- Ãgy minden rÃ©szproblÃ©mÃ¡t csak egyszer fogunk megoldani

PÃ©nzvÃ¡ltÃ¡si feladat megoldÃ¡sa DP-vel: minden Ã¶sszegre F-ig kiszÃ¡moljuk, hogy azt minimum hÃ¡ny pÃ©nzÃ©rmÃ©vel tudjuk kifizetni

- Ã¶tlet: minden Ã©rmÃ©re megnÃ©zzÃ¼k, hogy a korÃ¡bbi optimÃ¡lis megoldÃ¡s a jÃ³, amiben nem volt benne az az Ã©rme, vagy az, ha benne van az Ã©rme
- futÃ¡sidÅ: $O(Fn)$

IteratÃv megoldÃ¡s: bottom-up Ã©pÃtkezÃ¼nk, Ã©s minden lehetsÃ©ges Ã©rtÃ©ket megnÃ©zÃ¼nk RekurzÃv megoldÃ¡s memorizÃ¡lÃ¡ssal: top-down Ã©pÃtkezÃ¼nk, Ã©s kulcs-Ã©rtÃ©k pÃ¡rokat nÃ©zÃ¼nk (csak akkor, ha nem kell minden rÃ©szmegoldÃ¡s)

MohÃ³ algoritmusok

RÃ©szfeladatra bontÃ¡s OptimalizÃ¡lÃ¡s

A mohÃ³ algoritmus minden lÃ©pÃ©sben az aktuÃ¡lisan optimÃ¡lisnak tÅ±nÅ megoldÃ¡st vÃ¡lasztja. Nem minden problÃ©mÃ¡ra adhatÃ³ mohÃ³ algoritmus! De ha igen, akkor az nagyon hatÃ©kony

RÃ©szproblÃ©mÃ¡ra bontÃ¡skor a cÃ©l: a mohÃ³ vÃ¡lasztÃ¡s egyetlen rÃ©szproblÃ©mÃ¡t eredmÃ©nyezzen, aminek az optimÃ¡lis megoldÃ¡sa a problÃ©ma optimÃ¡lis megoldÃ¡sa is egyben

MohÃ³ algoritmusok helyessÃ©ge: - fogalmazzuk meg a feladatot Ãºgy, hogy minden dÃ¶ntÃ©s hatÃ¡sÃ¡ra egy kisebb rÃ©szproblÃ©ma keletkezzen - bizonyÃtsuk be, hogy mindig van mohÃ³ vÃ¡lasztÃ¡si lehetÅsÃ©g, tehÃ¡t biztonsÃ¡gos - mohÃ³ vÃ¡lasztÃ¡ssal olyan rÃ©szproblÃ©ma keletkezik, amihez hozzÃ¡vÃ©ve a mohÃ³ vÃ¡lasztÃ¡st, az eredeti problÃ©ma optimÃ¡lis megoldÃ¡sÃ¡t kapjuk (optimÃ¡lis rÃ©szstruktÃºrÃ¡k)

Egy feladat optimÃ¡lis rÃ©szstruktÃºrÃ¡jÃº, ha a problÃ©ma egy opt. megoldÃ¡sa tartalmazza a rÃ©szfeladatok optimÃ¡lis megoldÃ¡sait is.

PÃ©ldÃ¡k

HÃ¡tizsÃ¡k problÃ©ma

- adott egy hÃ¡tizsÃ¡k kapacitÃ¡sa, Ã©s n tÃ¡rgy, mindegyik Ã©rtÃ©kkel Ã©s sÃºllyal megadva
- mekkora a legnagyobb Ã¶sszÃ©rtÃ©k, amit a hÃ¡tizsÃ¡kba tehetÃ¼nk?

TÃ¶redÃ©kes hÃ¡tizsÃ¡k problÃ©ma

- a tÃ¡rgyak feldarabolhatÃ³k
- de minden tÃ¡rgybÃ³l egy darab van

MohÃ³ algoritmus a tÃ¶redÃ©kes hÃ¡tizsÃ¡kra:

- szÃ¡moljuk ki minden tÃ¡rgyra az Ã©rtÃ©k/sÃºly arÃ¡nyt
- tegyÃ¼k a hÃ¡tizsÃ¡kba a legnagyobb ilyen arÃ¡nyÃº tÃ¡rgyat, az egÃ©szet ha belefÃ©r, vagy tÃ¶rjÃ¼k, ha nem

2. Elemi adatszerkezetek, binÃ¡ris keresÅfÃ¡k, hasÃtÃ³ tÃ¡blÃ¡zatok, grÃ¡fok Ã©s fÃ¡k szÃ¡mÃtÃ³gÃ©pes reprezentÃ¡ciÃ³ja

Elemi adatszerkezetek

tÃ¶mb, lÃ¡ncolt lista, sor, verem, grÃ¡f, map, kupac - sajÃ¡t vÃ©lemÃ©ny

Adatszerkezet

- adatok tÃ¡rolÃ¡sÃ¡ra Ã©s szervezÃ©sÃ©re szolgÃ¡lÃ³ mÃ³dszer
- lehetÅvÃ© teszi a hatÃ©kony hozzÃ¡fÃ©rÃ©st Ã©s mÃ³dosÃtÃ¡sokat

Leggyakoribb mÅ±veletek

- beszÃºr
- keres
- tÃ¶rÃ¶l
- min
- max
- elÅzÅ
- kÃ¶vetkezÅ

MegfelelÅ adatszerkezet kulcs az implementÃ¡ciÃ³ futÃ¡sidejÃ©hez!

ListÃ¡k

Az adatok lineÃ¡risan kÃ¶vetik egymÃ¡st. Egy Ã©rtÃ©k tÃ¶bbszÃ¶r is elÅfordulhat.

MÅ±veletek: Ã©rtÃ©k, Ã©rtÃ©kad, keres, beszÃºr, tÃ¶rÃ¶l

KÃ¶zvetlen elÃ©rÃ©s

- minden index kÃ¶zvetlen elÃ©rÃ©sÅ±, kÃ¶zvetlenÃ¼l ÃrhatÃ³/olvashatÃ³
- Ã©rtÃ©k: O(1), keres: O(n)
- beszÃºr Ã©s tÃ¶rÃ¶l esetÃ©n vÃ¡ltozik a mÃ©ret, Ã¡t kell mÃ¡solni az elemeket egy Ãºj helyre
- beszÃºr: O(n), tÃ¶rÃ¶l: O(n)
- Ã¶tlet: ha tele van a tÃ¶mb, duplÃ¡zzuk meg a kapacitÃ¡st
- ha negyedÃ©re csÃ¶kken, felezzÃ¼k a kapacitÃ¡st
- Ãgy nem kell mindig az egÃ©sz tÃ¶mbÃ¶t mÃ¡solni

LÃ¡ncolt lista

minden Ã©rtÃ©k mellÃ© mutatÃ³kat tÃ¡rolunk a kÃ¶vetkezÅ/megelÅzÅ elemre

- egyszeresen lÃ¡ncolt: kÃ¶vetkezÅ elemre mutat
- kÃ©tszeresen lÃ¡ncolt: elÅzÅ Ã©s kÃ¶vetkezÅre is mutat
- ciklikus: az utolsÃ³ az elsÅ elemre mutat
- Årszem: egy nil elem, ami a lista elejÃ©re (fej) mutat

KÃ¶zvetlen elÃ©rÃ©s vs LÃ¡ncolt lista

- KÃ: Ã©rtÃ©k konstans, mÃ³dosÃtÃ³ lassÃº
- LL: Ã©rtÃ©k lassÃº, mÃ³dosÃtÃ³ gyors, sok memÃ³ria kell a mutatÃ³knak

Verem Ã©s sor

Stack, Queue

Olyan listÃ¡k, ahol a beszÃºrÃ¡s Ã©s a tÃ¶rlÃ©s csak adott pozÃciÃ³n tÃ¶rtÃ©nhet

- verem: legutoljÃ¡ra beszÃºrt elemet vehetjÃ¼k csak ki (LIFO - Last In First Out)
- sor: legkorÃ¡bban beszÃºrt elemet vehetjÃ¼k csak ki (FIFO - First In First Out)

Verem mÅ±veletek

- push: verem tetejÃ©re rakunk egy elemet
- pop: verem tetejÃ©rÅl levesszÃ¼k

Sor mÅ±veletek:

- enqueue: sor vÃ©gÃ©re rakunk
- dequeue: sor elejÃ©rÅl elveszÃ¼nk

PrioritÃ¡si sor Ã©s kupac

PrioritÃ¡si sor

- Ã©rkezÃ©s sorrendje lÃ©nyegtelen, mindig a min/max elemet akarjuk kivenni

lehet mondjuk listÃ¡val megvalÃ³sÃtani, veremmel vagy sorral nem Ã©rdemes, mert nem szÃ¡mÃt a sorrend

PrioritÃ¡si sor hatÃ©kony megvalÃ³sÃtÃ¡sa: kupac (heap)

Kupac

- majdnem teljes binÃ¡ris fa, minden csÃºcsa legalÃ¡bb akkora, mint a gyerekei -> max elem a gyÃ¶kÃ©rben

BinÃ¡ris keresÅfÃ¡k

Keres, beszÃºr, tÃ¶rÃ¶l, min, max, kÃ¶vetkezÅ, elÅzÅ Mind legyen O(logn)

BinÃ¡ris keresÅfa

- minden csÃºcsnak max kÃ©t gyereke van
- balra vannak a kisebb elemek
- jobbra a nagyobbak
- keresÃ©s O(h) idejÅ± (h a fa magassÃ¡ga)
- min/max is O(h): vagy teljesen jobbra, vagy teljesen balra kell lemennÃ¼nk
- kÃ¶vetkezÅ/elÅzÅ szintÃ©n O(h) - amÃg jobb/bal gyerek, addig megyÃ¼nk max
- beszÃºr szintÃ©n O(h) - mindig levÃ©lkÃ©nt szÃºrunk be, Ãºgy, hogy kb megkeressÃ¼k a helyÃ©t
- tÃ¶rÃ¶l is O(h), levelet simÃ¡n tÃ¶rlÃ¼nk, egy gyerekeset Ãºgy, hogy a gyereket linkeljÃ¼k a szÃ¼lÅhÃ¶z, kÃ©t gyerekeset pedig a kÃ¶vetkezÅvel helyettesÃtjÃ¼k

HasÃtÃ³ tÃ¡blÃ¡k

Halmaz Ã©s szÃ³tÃ¡r

Halmaz

- egy elem legfeljebb egyszer szerepelhet benne

- keres helyett tartalmaz-e
- beszÃºr, tÃ¶rÃ¶l
- metszet, uniÃ³

SzÃ³tÃ¡r

- kulcs Ã©rtÃ©k pÃ¡rok halmaza
- minden kulcs legfeljebb egyszer szerepelhet
- egy Ã©rtÃ©k tetszÅleges szÃ¡mban elÅfordulhat

AsszociatÃv tÃ¶mb

- egyÃ©szek helyett bÃ¡rmilyen tÃpussal indexelhetÃ¼nk

Map

- kulcs->Ã©rtÃ©k lekÃ©pezÃ©s

HasÃtÃ³tÃ¡blÃ¡k

Halmazok Ã©s szÃ³tÃ¡rak hatÃ©kony megvalÃ³sÃtÃ¡sa Keres, beszÃºr, tÃ¶rÃ¶l legyen hatÃ©kony

Ãtlagos esetben O(1)

HasÃtÃ³tÃ¡bla olyan szÃ³tÃ¡r, amikor egy hash fÃ¼ggvÃ©ny segÃtsÃ©gÃ©vel Ã¡llapÃtjuk meg, hogy melyik kulcshoz milyen Ã©rtÃ©k tartozzon

pÃ©lda: h(k) = k mod m ahol m a hasÃtÃ³ tÃ¡bla mÃ©rete lehetnek Ã¼tkÃ¶zÃ©sek! cÃ©l: az Ã¼tkÃ¶zÃ©sek minimalizÃ¡lÃ¡sa

GrÃ¡fok Ã©s fÃ¡k szÃ¡mÃtÃ³gÃ©pes reprezentÃ¡ciÃ³ja

SzomszÃ©dsÃ¡gi mÃ¡trix

- minden csÃºcshoz hozzÃ¡rendelÃ¼nk egy szÃ¡mot
- ha a Ã©s b kÃ¶zÃ¶tt van Ã©l, akkor matrix[a][b] = 1 Ã©s matrix[b][a] = 1
- ha nincs, akkor 0

SzomszÃ©dsÃ¡gi lista

- minden listaelem egy csÃºcs, ami szintÃ©n egy lista
- minden csÃºcshoz tartozÃ³ listÃ¡ban tÃ¡roljuk a vele szomszÃ©dos csÃºcsokat

Bal gyerek, jobb testvÃ©r

Bal gyerek, jobb gyerek

Binary Search Tree - tÃ¶mbbel is meg lehet

- Index of parent= INT[index of child node/2]
- Index of Left Child = 2 * Index of parent
- Index of Right Child = 2 * Index of parent+1

3. HatÃ©kony visszavezetÃ©s. Nemdeterminizmus. A P Ã©s NP osztÃ¡lyok. NP-teljes problÃ©mÃ¡k

HatÃ©kony visszavezetÃ©s

VisszavezetÃ©snek nevezzÃ¼k azt, mikor ha van egy problÃ©mÃ¡nk, amit nem tudjuk, hogy kÃ©ne megoldanunk, Ã©s egy problÃ©mÃ¡nk, amit tudjuk hogy oldjunk meg, Ã©s a nem ismert problÃ©ma inputjaibÃ³l elkÃ©szÃtjÃ¼k az ismert problÃ©ma egy inputjÃ¡t, Ã©s Ãgy oldjuk azt meg.

- Az Ã¡talakÃtÃ¡snak tartnaia kell a vÃ¡laszt
- Mindenre jÃ³ outputot kell adnia

HatÃ©konynak akkor nevezhetjÃ¼k, ha ez az *inputkonverziÃ³* polinomidejÅ±. Ezt Turing-visszavezetÃ©snek is hÃvjÃ¡k.

Nemdeterminizmus

Egy algoritmus *nemdeterminisztikus*, ha egy ponton Ãºgymond szÃ©tvÃ¡lik a futÃ¡sa, Ã©s tÃ¶bbfÃ©le eredmÃ©nye is lehet a futÃ¡s vÃ©gÃ©re.

P Ã©s NP osztÃ¡lyok

A *P* osztÃ¡lyban azok a problÃ©mÃ¡k vannak, amelyek determinisztikusan polinomidÅben megoldhatÃ³k.

Az *NP* osztÃ¡lyban azok a problÃ©mÃ¡k vannak, amelyek nemdeterminisztikusan polinomidÅben megoldhatÃ³k.

NP teljes problÃ©mÃ¡k

Egy problÃ©ma akkor *NP*-teljes, ha *NP*-beli Ã©s *NP*-nehÃ©z.

- *NP*-beli, ha nemdeterminisztikusan tudunk tanÃºkat generÃ¡lni hozzÃ¡, amik igen pÃ©ldÃ¡nyai a problÃ©mÃ¡nak
- *NP*-nehÃ©z, ha minden mÃ¡s *NP*-beli problÃ©mÃ¡t hatÃ©konyan vissza tudunk vezetni rÃ¡.

PÃ©ldÃ¡k

SAT, HÃ¡tizsÃ¡k, Hamilton-Ãºt, Hamilton-kÃ¶r, Euler-kÃ¶r, ILP, RÃ©szletÃ¶sszeg, PartÃciÃ³

4. A PSPACE osztÃ¡ly. PSPACE-teljes problÃ©mÃ¡k. Logaritmikus tÃ¡rigÃ©nyÅ± visszavezetÃ©s. NL-teljes problÃ©mÃ¡k

PSPACE osztÃ¡ly

Savitch-tÃ©tel

- ElÃ©rhetÅsÃ©g eldÃ¶nthetÅ O(log^2n) tÃ¡rban

Az f(n) tÃ¡rban nemdeterminisztikusan eldÃ¶nthetÅ problÃ©mÃ¡k mind eldÃ¶nthetÅk determinisztikusan, f^2(n) tÃ¡rban is

TehÃ¡t: NSPACE(f(n)) rÃ©szhalmaza SPACE(f^2(n))-nek Ã©s mivel polinom nÃ©gyzete polinom PSPACE = NPSPACE

Polinom tÃ¡rban (det. vagy nemdet.) eldÃ¶nthetÅ problÃ©mÃ¡k osztÃ¡lya

PSPACE-teljes problÃ©mÃ¡k

QSAT PSPACE-teljes

QSAT (kvantifikÃ¡lt SAT)

- adott egy ÃtÃ©letkalkulusbeli logikai formula, vÃ¡ltozÃ³ kvantorokkal az elejÃ©n (lÃ©tezik, bÃ¡rmely, lÃ©tezik, bÃ¡rmely stb)
- magja CNF alakÃº, kvantormentes
- igaz-e ez a formula?

QSAT mint kÃ©tszemÃ©lyes jÃ¡tÃ©k

- input ugyanaz
- van-e az elsÅ jÃ¡tÃ©kosnak nyerÅ stratÃ©giÃ¡ja abban a jÃ¡tÃ©kban, ahol:
-
    - a jÃ¡tÃ©kosok sorban Ã©rtÃ©ket adnak a vÃ¡ltozÃ³knak, elsÅ jÃ¡tÃ©kos x1-nek, mÃ¡sodik x2-nek stb
-
    - ha a formula igaz lesz, az elsÅ jÃ¡tÃ©kos nyert, ha hamis, akkor a mÃ¡sodik
- ez ugyanaz tkp, mint a sima QSAT, szÃ³val ez is PSPACE-teljes

hasonlÃt a minimaxra az Ã©ses csÃºcsoknÃ¡l lÃ©vÅ jÃ¡tÃ©kos minimalizÃ¡l

FÃ¶ldrajzi jÃ¡tÃ©k

- adott egy irÃ¡nyÃtott grÃ¡f Ã©s egy kijelÃ¶lt kezdÅcsÃºcs
- az elsÅ jÃ¡tÃ©kosnak van-e nyerÅ stratÃ©giÃ¡ja?

-
    - az elsÅ jÃ¡tÃ©kos kezd, lerakja a bÃ¡but a kezdÅcsÃºcsra
-
    - felvÃ¡ltva lÃ©pnek
-
    - egy olyan csÃºcsba kell hÃºzni a bÃ¡but, ami egy lÃ©pÃ©sben elÃ©rhetÅ, Ã©s ahol mÃ©g nem voltak
-
    - aki elÅszÃ¶r nem tud lÃ©pni, vesztett

FÃ¶ldrajzi jÃ¡tÃ©k PSPACE-teljes

Adott kÃ©t regulÃ¡ris kifejezÃ©s, igaz-e, hogy ugyanazokra a szavakra illeszkednek? Adott kÃ©t nemdet automata, ekvivalensek-e? Adott, egy SOKOBAN/RUSH HOUR feladvÃ¡ny, megoldhatÃ³-e?

LogtÃ¡ras visszavezetÃ©s

PolinomidejÅ± visszavezetÃ©s tÃºl erÅs, ha pl P-beli problÃ©mÃ¡kat akarunk egymÃ¡shoz viszonyÃtani, mert egy polinomidejÅ± visszavezetÃ©s alatt mÃ¡r akÃ¡r meg is oldhatnÃ¡nk egy P-beli problÃ©mÃ¡t

LogtÃ¡ras visszavezetÃ©s

f egy olyan fÃ¼ggvÃ©ny, hogy

- A inputjaibÃ³l B inputjait kÃ©szÃti
- vÃ¡lasztartÃ³ mÃ³don
- Ã©s logaritmikus tÃ¡rban kiszÃ¡mÃthatÃ³

akkor f egy logtÃ¡ras visszavezetÃ©s A-rÃ³l B-re.

5. VÃ©ges automata Ã©s vÃ¡ltozatai, a felismert nyelv definÃciÃ³ja. A regulÃ¡ris nyelvtanok, a vÃ©ges automatÃ¡k Ã©s a regulÃ¡ris kifejezÃ©sek ekvivalenciÃ¡ja. RegulÃ¡ris nyelvekre vonatkozÃ³ pumpÃ¡lÃ³ lemma, alkalmazÃ¡sa Ã©s kÃ¶vetkezmÃ©nyei

VÃ©ges automata Ã©s vÃ¡ltozatai, a felismert nyelv definÃciÃ³ja

LÃ¡sd pdf

A regulÃ¡ris nyelvtanok, a vÃ©ges automatÃ¡k Ã©s a regulÃ¡ris kifejezÃ©sek ekvivalenciÃ¡ja

RegulÃ¡ris nyelvtanok

- N: nemterminÃ¡lis abc
- SZIGMA: terminÃ¡lis abc
- P: szabÃ¡lyok halmaza

- S: eleme N, kezdÅ nemterminÃ¡lis Egy G=(N, SZIGMA, P, S) nyelvtan regulÃ¡ris (vagy jobblineÃ¡ris), ha P-ben minden szabÃ¡ly

- A -> xB vagy

- A -> x

alakÃº.

AzÃ©rt jobblineÃ¡ris, mert minden szabÃ¡ly jobb oldalÃ¡n max. egy nemterminÃ¡lis Ã¡llhat, Ã©s ez mindig a szÃ³ vÃ©gÃ©n helyezkedik el. LevezetÃ©st csak A -> x alakÃº szabÃ¡llyal fejezhetÃ¼nk be, ahol x eleme SZIGMA*. A regulÃ¡ris nyelvtanok speciÃ¡lis kÃ¶rnyezetfÃ¼ggetlen nyelvtanok.

PÃ©lda: S â aaS|abS|baS|bbS|Îµ, vagyis a pÃ¡ros hosszÃº szavakat generÃ¡lÃ³ nyelvtan.

RegulÃ¡ris kifejezÃ©sek

VeszÃ¼nk egy abc-t, Ã©s hozzÃ¡veszÃ¼nk nÃ©hÃ¡ny szimbÃ³lumot, ezekbÅl Ã©pÃtÃ¼nk regulÃ¡ris kifejezÃ©seket.

A szigma feletti regulÃ¡ris kifejezÃ©sek halmaza a (Î£âª{â, Îµ,(,),+,â})* halmaz legszÅ±kebb olyan U rÃ©szhalmaza, hogy

- â, Îµ eleme U-nak
- minden a eleme Î£ eleme U-nak
- ha R1, R2 eleme U, akkor R1+R2, R1R2, R1* is eleme U-nak

PrioritÃ¡si sorrend: *, konkatenÃ¡ciÃ³, +

JelentÃ©sek:

- |R|, az R Ã¡ltal reprezentÃ¡lt nyelv
- R = â, |R| = â, azaz az Ã¼res nyelv
- R = Îµ, |R| = {Îµ}, azaz az epszilon szimbÃ³lum Ã¶nmagÃ¡ban, mint nyelv
- R = a, |R| = {a}, azaz az a szimbÃ³lum Ã¶nmagÃ¡ban, mint nyelv
- R = R1+R2, |R| = |R1| âª |R2|, azaz a kÃ©t regex Ã¡ltal generÃ¡lt nyelv uniÃ³ja
- R = R1R2, |R| = |R1||R2|, azaz a kÃ©t regex Ã¡ltal generÃ¡lt nyelv konkatenÃ¡ciÃ³ja
- R = R1*, akkor |R| = |R1*|*, azaz a regex Ã¡ltal generÃ¡lt nyelv iterÃ¡ciÃ³ja, az Ã¶sszes szÃ³ Ã¶sszekonkatenÃ¡lva egy mÃ¡sik nyelvbeli szÃ³val az Ã¶sszes lehetsÃ©ges mÃ³don

Ekvivalencia

TetszÅleges L rÃ©szhalmaza szigmacsillag nyelv esetÃ©n a kÃ¶vetkezÅ hÃ¡rom Ã¡llÃtÃ¡s ekvivalens:

- 1. L generÃ¡lhatÃ³ regulÃ¡ris nyelvtannal
- 1. L felismerhetÅ automatÃ¡val
- 1. L reprezentÃ¡lhatÃ³ regulÃ¡ris kifejezÃ©ssel

3 -> 1

Ha L reprezentÃ¡lhatÃ³ regulÃ¡ris kifejezÃ©ssel, akkor generÃ¡lhatÃ³ regulÃ¡ris nyelvtannal.

BizonyÃtÃ¡s: L-et reprezentÃ¡lÃ³ R regulÃ¡ris kifejezÃ©s struktÃºrÃ¡ja szerinti indukciÃ³val.

- R = â: ekkor L=|R|=â, ekkor L generÃ¡lhatÃ³ a G=(N, SZIGMA, â, S) nyelvtannal.
- R = a: a eleme SZIGMA, vagy a=epszilon, ekkor L=|R|={a}, ami generÃ¡lhatÃ³ a G=(N, SZIGMA, {S -> A}, S) nyelvtannal
- INDUKCIÃ R = R1+R2, ekkor L=|R|=L1 uniÃ³ L2, L1=|R1|, L2=|R2|
  - ekkor tegyÃ¼k fel, hogy Li generÃ¡lhatÃ³ egy Gi= (Ni,Î£,Pi,Si) (i=1,2) regulÃ¡ris nyelvtannal
  - ekkor L generÃ¡lhatÃ³ egy G= (N1âªN2âª{S},Î£,P1âªP2âª{SâS1,SâS2},S) nyelvtannal, ahol S egy Ãºj szimbÃ³lum
  - aka vesszÃ¼k az Ã¶sszes nemterminÃ¡list, az abc-t, az Ã¶sszes korÃ¡bbi szabÃ¡lyt
  - tovÃ¡bbÃ¡ egy Ãºj kezdÅszimbÃ³lumot
  - Ã©s az Ãºj kezdÅszimbÃ³lumbÃ³l elÃ©rhetÅ a rÃ©gi kettÅ kezdÅszimbÃ³lum, aka kivÃ¡laszthatjuk, melyik nyelvbÅl szÃ¡rmazÃ³ szÃ³t akarjuk generÃ¡lni
  - ahhoz, hogy elÃ©rhetÅ legyen a rÃ©gi kÃ©t kezdÅszimbÃ³lum, felveszÃ¼nk kÃ©t Ãºj szabÃ¡lyt Ã©rtelemszerÅ±en a rÃ©giek kÃ¶zÃ©
- INDUKCIÃ R = R1R2, ekkor L=|R|=L1L2, L1=|R1|, L2=|R2|
  - ekkor tegyÃ¼k fel, hogy Li generÃ¡lhatÃ³ egy Gi= (Ni,Î£,Pi,Si) (i=1,2) regulÃ¡ris nyelvtannal
  - Akkor L generÃ¡lhatÃ³ a G = (N1âªN2,Î£,P,S1) nyelvtannal, ahol P:
    - belevesszÃ¼k az Ã¶sszes szabÃ¡lyt az elsÅ nyelvet generÃ¡lÃ³ nyelvtanbÃ³l, a befejezÅ szabÃ¡lyok vÃ©gÃ©re odaÃrjuk a mÃ¡sodik nyelvtan kezdÅszimbÃ³lumÃ¡t
    - a mÃ¡sodik nyelvtan Ã¶sszes szabÃ¡lyÃ¡t is belevesszÃ¼k
- INDUKCIÃ R = R1*, ekkor L=|R|=L1*, ahol L1=|R1|
  - ekkor tegyÃ¼k fel, hogy L1 generÃ¡lhatÃ³ egy G1= (N1,Î£,P1,S1) nyelvtannal
  - ekkor L generÃ¡lhatÃ³ egy G= (N1âª{S},Î£,P,S) nyelvtannal, ahol S egy Ãºj szimbÃ³lum
  - a szabÃ¡lyokat Ãºgy mÃ³dosÃtjuk, hogy:
    - S-bÅl elÃ©rhetÅ az Ã¼resszÃ³ Ã©s az eredeti kezdÅszimbÃ³lum
    - a nem "befejezÅ" szabÃ¡lyokat felvesszÃ¼k Ãºgy, ahogy voltak
    - a "befejezÅ" szabÃ¡lyok jobb oldalÃ¡ra odaÃrjuk az S-t

1 -> 2

Ha L nyelv regulÃ¡ris, akkor felismerhetÅ automatÃ¡val.

BizonyÃtÃ¡s: legyen L egy regulÃ¡ris nyelv, Ã©s L=L(G), ahol G egy regulÃ¡ris nyelvtan.

Minden G= (N,Î£,P,S), regulÃ¡ris nyelvtanhoz megadhatÃ³ vele ekvivalens Gâ²= (Nâ²,Î£,Pâ²,S) regulÃ¡ris nyelvtan, Ìugy hogy Pâ²-ben minden szabÃ¡ly AâB, AâaB vagy AâÎµ alakÃº, ahol A,B â N Ìes a â Î£.

BizonyÃtÃ¡s

- amelyik szabÃ¡ly mÃ¡r alapbÃ³l ilyen alakÃº, azokat felvesszÃ¼k P'-be
- az A -> a1a2a3...anB alakÃº szabÃ¡lyokat szÃ©tdaraboljuk
  - lesz belÅle A -> a1A1, A1 -> a2A2 -> ... An-1 -> anB
- az A -> a1a2a3...an szabÃ¡lyokat feladarboljuk
  - lesz belÅle A -> a1A1, A1 -> a2A2 -> ... -> An -> epszilon

az Ãºj nemterminÃ¡lisokat felvesszÃ¼k N-be

Minden olyan G= (N,Î£,P,S) regulÃ¡ris nyelvtanhoz, melynek csak AâB, AâaB vagy AâÎµ alakÃº szabÃ¡lyai vannak megadhatÃ³ olyan M= (Q,Î£, Î´,q0,F) nemdeterminisztkius Îµ-automata, amelyre L(M) =L(G).

BizonyÃtÃ¡s

- Q = N, azaz a nemterminÃ¡lisokbÃ³l lesznek az Ã¡llapotok
- q0 = S, a kezdÅszimbÃ³lumbÃ³l lesz a kezdÅÃ¡llapot
- azokbÃ³l a B nemterminÃ¡lisokbÃ³l lesz vÃ©gÃ¡llapot, amikbÅl van B -> epszilon szabÃ¡ly
- minden A -> aB kinÃ©zetÅ± szabÃ¡lybÃ³l pedig legyen egy Ã¡tmenet A-bÃ³l B-be a hatÃ¡sÃ¡ra

2 -> 3

Minden automatÃ¡val felismert nyelv reprezentÃ¡lhatÃ³ regulÃ¡ris kifejezÃ©ssel. (Kleene tÃ©tele)

BizonyÃtÃ¡s: legyen L=(M), ahol M egy determinisztikus automata. Megadunk egy olyan regulÃ¡ris kifejezÃ©st, ami L-et reprezentÃ¡lja.

TÃ©telezzÃ¼k fel, hogy Q={1,2,3,...,n}, Ã©s q0=1. Minden 0 kisebbegyenlÅ k, azaz k darab Ã¡llapot, Ã©s 0 kisebbegyenlÅ i, j kisebbegyenlÅ n esetÃ©n definiÃ¡ljuk az L^(k)_i,j nyelvet a kÃ¶vetkezÅkÃ©ppen:

NÃ©zzÃ¼k Ãºgy az automatÃ¡t, mintha az i. Ã¡llapotbÃ³l indulnÃ¡nk, Ã©s a j.-be akarnÃ¡nk eljutni, de csak az {1,...,k} Ã¡llapotokat Ã©rinthetjÃ¼k. Az L^(k)_i,j nyelv azokat a szavakat tartalmazza, amelyeket ez a "kisebb" automata felismer.

EzutÃ¡n vegyÃ¼k Ã©szre azt, hogy az L nyelv tulajdonkÃ©ppen Ãºgy Ã¡ll elÅ, hogy venni kell az Ã¶sszes olyan L^(n)_1,j nyelvet, ahol j egy vÃ©gÃ¡llapot! TehÃ¡t vegyÃ¼k az Ã¶sszes olyan nyelvet, ahol az elsÅ Ã¡llapotbÃ³l akarunk elindulni, Ã©s az utolsÃ³ba eljutni, Ã©s hasznÃ¡lhatjuk az Ã¶sszes Ã¡llapotÃ¡t M-nek (mind az n darabot). TehÃ¡t ezen L^(n)_1,j nyelvek uniÃ³ja lesz L.

EzÃ©rt elÃ©g az L^(n)_1,j-ket reprezentÃ¡lÃ³ regulÃ¡ris kifejezÃ©seket megadnunk (R^(n)_1,j).

Ehhez meg kell adnunk az R^(k)_i,j regulÃ¡ris kifejezÃ©seket k szerinti indukciÃ³val.

k=0 azt jelenti, hogy 0 kÃ¶zbÃ¼lsÅ Ã¡llapotbÃ³l kell eljutnunk az i Ã¡llapotbÃ³l a j Ã¡llapotba. Ez lehet Ãºgy, hogy valamilyen szimbÃ³lum hatÃ¡sÃ¡ra Ã¡tmegyÃ¼nk, vagy ha i=j, akkor hurokÃ©llel helyben maradunk, vagy epszilonnal nem csinÃ¡lunk semmit.

Az indukciÃ³s feltevÃ©sÃ¼nk az, hogy minden i,j-re megadtuk az R^(k)_i,j-t

k+1-hez ÃSZREVESSZÃK (ja ugye tÃ¶k triviÃ¡lis lmao)

L^(k+1)_i,j egyenlÅ azzal, hogy

- vagy amÃºgyis eljutunk k kÃ¶ztes Ã¡llapottal is i-bÅl j-be
- vagy belevesszÃ¼k a k+1. Ã¡llapotot is a levesbe, elmegyÃ¼nk az 1-estÅl a k+1. Ã¡llapotba, ott kÃ¶rÃ¶zÃ¼nk akÃ¡rmeddig, Ã©s utÃ¡na k+1-bÅl pedig eljutunk j-be

Ekkor, mivel az L^(k)_i,j nyelvekhez az indukciÃ³s feltevÃ©s miatt tudtunk megfelelÅ regexet adni, ezekre elvÃ©gezve az elÅzÅ azonossÃ¡got, megkapjuk az R^(k+1)_i,j-t is, ezzel pedig meg tudjuk kapni az Ã¶sszes regexet, ami a kezdÅÃ¡llapotbÃ³l a vÃ©gÃ¡llapotokba visz, ezeket uniÃ³zva pedig az egÃ©sz nyelvhez tartozÃ³ regexet.

RegulÃ¡ris nyelvekre vonatkozÃ³ pumpÃ¡lÃ³ lemma, alkalmazÃ¡sa Ã©s kÃ¶vetkezmÃ©nyei

PumpÃ¡lÃ³ lemma

Minden L regulÃ¡ris nyelv esetÃ©n megadhatÃ³ egy L-tÅl fÃ¼ggÅ k>0 szÃ¡m, melyre minden w eleme L esetÃ©n, ha |w| nagyobbegyenlÅ k, akkor van olyan w = w1w2w3 felbontÃ¡s, hogy

- 0 < |w2| Ã©s |w1w2| kisebbegyenlÅ k
- minden n nagyobbegyenlÅ 0-ra w1w2^nw3 eleme L

FordÃtva, ha egy nyelvhez nem adhatÃ³ meg ilyen k szÃ¡m, akkor a nyelv nem regulÃ¡ris.

Kb a lÃ©nyeg, hogy ha egy nyelv regulÃ¡ris, akkor a k-nÃ¡l hosszabb szavak felbonthatÃ³k hÃ¡rom rÃ©szre, Ã©s a kÃ¶zÃ©psÅ rÃ©sz ismÃ©tlÅdhet akÃ¡rmeddig

AlkalmazÃ¡sa

Pl bebizonyÃthatjuk vele egy nyelvrÅl, hogy nem regulÃ¡ris. a^nb^n n >= 0 nyelv nem regulÃ¡ris

tegyÃ¼k fel, hogy van ilyen k, amivel felbonthatÃ³ a feltÃ©telek miatt w2-ben csak a betÅ±k vannak ha ezt pumpÃ¡ljuk, tÃ¶bb a betÅ± lesz benne, mint b - rossz

KÃ¶vetkezmÃ©nyei

Vannak olyan nyelvek, amelyek nem kÃ¶rnyezetfÃ¼ggetlenek, de nem regulÃ¡risak, pl a^nb^n n >= 0.

6. A kÃ¶rnyezetfÃ¼ggetlen nyelvtan Ã©s nyelv definÃciÃ³ja. DerivÃ¡ciÃ³k Ã©s derivÃ¡ciÃ³s fÃ¡k kapcsolata. VeremautomatÃ¡k Ã©s kÃ¶rnyezetfÃ¼ggetlen nyelvtanok ekvivalenciÃ¡ja. A Bar-Hillel lemma Ã©s alkalmazÃ¡sa

A kÃ¶rnyezetfÃ¼ggetlen nyelvtan Ã©s nyelv definÃciÃ³ja

Egy G=(N, SZIGMA, P, S) nyelvtan, kÃ¶rnyezetfÃ¼ggetlen, ha minden szabÃ¡lya A -> alfa alakÃº, ahol alfa egy terminÃ¡lisokbÃ³l Ã©s nemterminÃ¡lisokbÃ³l Ã¡llÃ³ szÃ³.

Egy nyelv kÃ¶rnyezetfÃ¼ggetlen, ha van olyan CF nyelvtan, ami Åt generÃ¡lja.

DerivÃ¡ciÃ³k Ã©s derivÃ¡ciÃ³s fÃ¡k kapcsolata

KorlÃ¡tozÃ¡s nÃ©lkÃ¼li derivÃ¡ciÃ³

- bÃ¡rmely nemterminÃ¡lis helyÃ©re helyettesÃthetÃ¼nk

Bal/jobboldali derivÃ¡ciÃ³

- csak a legbal/jobboldalabbi nemterminÃ¡lisba helyettesÃthetÃ¼nk

DerivÃ¡ciÃ³s fÃ¡k

Mindig csak egy gyÃ¶kere van

- vagy csak a gyÃ¶kÃ©rbÅl Ã¡ll
- vagy van egy epszilon gyerek
- vagy kiindul belÅle k darab Ã©l, amelyek vÃ©gpontjai tovÃ¡bbi derivÃ¡ciÃ³s fÃ¡k gyÃ¶kerei

Legyen t egy derivÃ¡ciÃ³s fa, gyÃ¶kere X t magassÃ¡ga h(t) t hatÃ¡ra fr(t) - hatÃ¡r kb a levelek balrÃ³l jobbra olvasva

DerivÃ¡ciÃ³s fÃ¡k kapcsolata a derivÃ¡ciÃ³kkal

TetszÅleges X gyÃ¶kerÅ± derivÃ¡ciÃ³s fÃ¡ra Ã©s alfa szÃ³ra X-bÅl akkor vezethetÅ le alfa, ha van olyan X gyÃ¶kerÅ± derivÃ¡ciÃ³s fa, amelyre fr(t) = alfa

VeremautomatÃ¡k Ã©s kÃ¶rnyezetfÃ¼ggetlen nyelvtanok ekvivalenciÃ¡ja

Veremautomata

VeremautomatÃ¡nak nevezzÃ¼k azt a P= (Q,Î£,Î,Î´,q0,Z0,F), ahol

- Q: Ã¡llapotok halmaza
- Î£: input abc
- Î: verem abc
- q0 eleme Q: kezdÅÃ¡llapot
- Z0: verem kezdÅszimbÃ³lum
- F: vÃ©gÃ¡llapotok halmaza
- Î´: Ã¡tmenetfÃ¼ggvÃ©ny

Az Ã¡tmenet a kÃ¶vetkezÅkÃ©ppen tÃ¶rtÃ©nik: ha az automata a q Ã¡llapotban van, a szimbÃ³lum Ã©rkezik Ã©s Z van a verem tetejÃ©n, akkor Ã¡tmegy a q_i Ã¡llapotba, a veremben pedig Z helyÃ©re alfa_i kerÃ¼l. Az Ã¡tmenetnÃ©l az automata kiolvas egy betÅ±t az inputbÃ³l, leveszi Z-t a verem tetejÃ©rÅl, Ã©s tetszÅleges hosszÃº szÃ³t odaÃr a helyÃ©re.

Egy szÃ³ elfogadÃ¡sa tÃ¶rtÃ©nhet vÃ©gÃ¡llapotokkal, vagy Ã¼res veremmel is. Ugyanazon automatÃ¡nÃ¡l Ã¡ltalÃ¡ban nem egyezik meg az Ã¼res veremmel Ã©s a vÃ©gÃ¡llapotokkal felismert nyelv.

Ekvivalencia

Minden kÃ¶rnyezetfÃ¼ggetlen nyelvtanhoz meg lehet adni veremautomatÃ¡t Ãºgy, hogy a veremautomata (Ã¼res veremmel vagy vÃ©gÃ¡llapottal) ugyanazt a nyelvet ismeri fel, amit a kÃ¶rnyezetfÃ¼ggetlen nyelvtan generÃ¡l.

A bizonyÃtÃ¡shoz egy kÃ¶rnyezetfÃ¼ggetlen nyelvtanhoz konstruÃ¡lunk egy egyÃ¡llapotos nemdeterminisztikus veremautomatÃ¡t, ami Ã¼res veremmel ismeri fel a szavakat. A veremabc legyen a nemterminÃ¡lisok uniÃ³ terminÃ¡lisok. Ezzel a veremautomatÃ¡val szimulÃ¡ljuk a nyelvtan levezetÃ©seit. Tudjuk tovÃ¡bbÃ¡, hogy az Ã¼res veremmel Ã©s a vÃ©gÃ¡llapotokkal felismert nyelvek halmaza ugyanaz, Ãgy ez az Ã¡llÃtÃ¡sunk igaz lesz.

Minden veremautomatÃ¡val felismert nyelv kÃ¶rnyezetfÃ¼ggetlen.

Itt pedig veremautomatÃ¡hoz adunk meg egy kÃ¶rnyezetfÃ¼ggetlen nyelvtant.

LÃ¡sd pdf 2.

Bar-Hillel lemma Ã©s alkalmazÃ¡sa

TulajdonkÃ©ppen pumpÃ¡lÃ³ lemma CF nyelvekre

Ha L egy kÃ¶rnyezetfÃ¼ggetlen nyelv, akkor lÃ©tezik egy nyelvtÅl fÃ¼ggÅ k szÃ¡m, amire ha egy L-beli szÃ³ hossza nagyobb k-nÃ¡l, akkor feldarabolhatÃ³ 5 rÃ©szre, amikre a kÃ¶vetkezÅk teljesÃ¼lnek:

- |w2w3w4| <= k
- w2w4 nem epszilon
- minden n >= 0-ra w1w2^nw3w4^nw5 eleme L-nek

AlkalmazÃ¡s: az L={a^nb^nc^n|nâ¥1} nyelv nem kÃ¶rnyezetfÃ¼ggetlen.

TegyÃ¼k fel, hogy igen, ekkor lÃ©teznie kell olyan k szÃ¡mnak, amire teljesÃ¼lnek a Bar-Hillel lemmÃ¡ban a feltÃ©telek.

VegyÃ¼k az a^kb^kc^k szÃ³t, aminek hossza 3k >= k, tehÃ¡t jÃ³ lesz fixen.

A lemma szerint ennek lÃ©tezik w1w2w3w4w5 felbontÃ¡sa, melyre w2w4 nem epszilon, Ã©s minden n >= 0-ra w1w2^nw3w4^nw5 eleme a nyelvnek.

NÃ©zzÃ¼k ekkor mi lehet w2-ben Ã©s w4-ben! Egyik sem tartalmazhat kÃ©t betÅ±t, mert ekkor pl ha kÃ©tszer vesszÃ¼k w2-t Ã©s w4-et, akkor a betÅ±k sorrendje nem abc lesz. TehÃ¡t biztosan csak egyfÃ©le betÅ±t tartalmaznak. Ekkor a w1w2^2w3w4^2w5 szÃ³bal legalÃ¡bb egy, Ã©s legfeljebb kÃ©t betÅ± szÃ¡ma tÃ¶bb, mint a tÃ¶bbi betÅ±Ã©, tehÃ¡t biztos nem eleme ez a szÃ³ L-nek.

7. EliminÃ¡ciÃ³s mÃ³dszerek, mÃ¡trixok triangulÃ¡ris felbontÃ¡sai. LineÃ¡ris egyenletrendszerek megoldÃ¡sa iterÃ¡ciÃ³s mÃ³dszerekkel. MÃ¡trixok sajÃ¡tÃ©rtÃ©keinek Ã©s sajÃ¡tvektorainak numerikus meghatÃ¡rozÃ¡sa

EliminÃ¡ciÃ³s mÃ³dszerek

Gauss-eliminÃ¡ciÃ³

- Ax=b alakÃº lineÃ¡ris egyenletrendszerek megoldÃ¡sÃ¡hoz tudjuk hasznÃ¡lni
- az Ax=b egyenletrendszernek pontosan akkor van egy megoldÃ¡sa, ha det(A) nem 0
- ekkor x = A^-1b
  - de az inverzet kiszÃ¡molni tÃºl lassÃº lenne

A Gauss-eliminÃ¡ciÃ³val az A mÃ¡trixot felsÅ hÃ¡romszÃ¶gmÃ¡trixszÃ¡ alakÃtjuk, Ã©s ha ez sikerÃ¼l, akkor abbÃ³l visszahelyettesÃ-tÃ©sekkel megkaphatjuk x-et. MÅ±veletigÃ©nye O(n^2/2).

A felsÅ hÃ¡romszÃ¶gmÃ¡trixot Ãºn. eliminÃ¡ciÃ³s mÃ¡trixok segÃtsÃ©gÃ©vel kapjuk meg. Egy eliminÃ¡ciÃ³s mÃ¡trix dolga, hogy kinullÃ¡zza az A mÃ¡trix egyik oszlopÃ¡ban a fÅÃ¡tlÃ³ alatti elemeket. Ha az Ã¶sszes ilyen eliminÃ¡ciÃ³s mÃ¡trixot Ã¶sszeszorozzuk balrÃ³l egymÃ¡ssal, akkor kapjuk az M mÃ¡trixot. Ekkor az MA szorzÃ¡s eredmÃ©nye lesz a kÃvÃ¡nt felsÅ triangulÃ¡ris mÃ¡trix.

LU felbontÃ¡s

Az LU felbontÃ¡s lÃ©nyege, hogy az A mÃ¡trixot egy alsÃ³ Ã©s egy felsÅ hÃ¡romszÃ¶gmÃ¡trixra bontjuk. A Gauss eliminÃ¡ciÃ³hoz nagyon hasonlÃt, ott az MA szorzÃ¡s eredmÃ©nye egy U felsÅ triangulÃ¡ris mÃ¡trix volt. Ha mindkÃ©t oldalt megszorozzuk balrÃ³l M^-1-gyel, akkor azt kapjuk, hogy A = M^-1U. Legyen M^-1=L, mert M^-1 egy alsÃ³ triangulÃ¡ris mÃ¡trix. Ezzel elvÃ©geztÃ¼k az A mÃ¡trix LU felbontÃ¡sÃ¡t.

Ekkor az Ax=b egyeletrendszer megoldÃ¡sÃ¡t a kÃ¶vetkezÅkÃ©ppen kaphatjuk:

- Ax=b
- LUx=b
- y = Ux
- Ly=b
- y = L^-1b
- L^-1b = Ux

Cholesky felbontÃ¡s

Ha az A mÃ¡trix

- szimmetrikus
- pozitÃv definit
  - minden sajÃ¡tÃ©rtÃ©ke pozitÃv

akkor felbonthatÃ³ a kÃ¶vetkezÅkÃ©ppen: A=LL^T, tehÃ¡t U az most pont L transzponÃ¡ltja lesz.

QR felbontÃ¡s

Ha az A mÃ¡trix

- nÃ©gyzetes
- valÃ³s
- regulÃ¡ris (det(A) nem 0)

akkor lÃ©tezik QR felbontÃ¡sa.

Q egy Ãºgynevezett ortogonÃ¡lis mÃ¡trix (Ã©s nÃ©gyzetes is). Ez azt jelenti, hogy Q^TQ = QQ^T = I. R egy felsÅ hÃ¡romszÃ¶gmÃ¡trix

BizonyÃtÃ¡s: A^TA pozitÃv definit, Ãgy lÃ©tezik R^TR Cholesky felbontÃ¡sa.

Legyen ekkor Q egyenlÅ A^R-1-gyel.

Igazoljuk, hogy Q ortogonÃ¡lis.

Q^TQ = (AR^-1)^T*(A^R^-1) = (R^-1)^T*A^T*A*R^-1 = (R^-1)^T*R^T*R*R^-1 = I*I = I behelyettesÃtÃ©s transzponÃ¡lÃ¡sos azonossÃ¡g A^TA=R^TR inverzek kiÃ¼tik egymÃ¡st

TehÃ¡t Q valÃ³ban ortogonÃ¡lis.

LineÃ¡ris egyenletrendszerek megoldÃ¡sa iterÃ¡ciÃ³s mÃ³dszerekkel

Jacobi iterÃ¡ciÃ³

ÃtrendezzÃ¼k Ãºgy az egyenletrendszert, hogy a baloldalon egy-egy vÃ¡ltozÃ³t kifejezÃ¼nk

VÃ¡lasztunk valami indulÃ³vektort, ami ilyen kezdÅ megoldÃ¡s kb A vektor elemeit behelyettesÃtjÃ¼k a jobboldalra, Ã©s ebbÅl kapunk egy

Ãºj vektort a baloldalon, ezzel folytatjuk.

Csak akkor konvergÃ¡l, ha a mÃ¡trix *szigorÃºan diagonÃ¡lisan dominÃ¡ns*, vagyis az Ã¶sszes fÅÃ¡tlÃ³beli elem abszolÃºt Ã©rtÃ©ke a legnagyobb az adott sorban.

Gauss-Sediel iterÃ¡ciÃ³

Ugyanaz, mint a Jacobi, csak ha mÃ¡r egy vÃ¡ltozÃ³ Ãºj Ã©rtÃ©kÃ©t kiszÃ¡moltuk, akkor a kÃ¶vetkezÅ sorokban mÃ¡r azt az Ãºj Ã©rtÃ©ket hasznÃ¡ljuk.

MÃ¡trixok sajÃ¡tÃ©rtÃ©keinek Ã©s sajÃ¡tvektorainak numerikus meghatÃ¡rozÃ¡sa

SajÃ¡tÃ©rtÃ©k, sajÃ¡tvektor

Ax = lambda x

x a sajÃ¡tvektor, lambda a sajÃ¡tÃ©rtÃ©k

A sajÃ¡tÃ©rtÃ©k olyan szÃ¡m, amivel ha megszorozzuk a hozzÃ¡ tartozÃ³ sajÃ¡tvektort, akkor ugyanazt az eredmÃ©nyt kapjuk, mintha azt a vektort a mÃ¡trixszal szoroztuk volna meg.

MeghatÃ¡rozÃ¡sa: det(A - lambdaI) = 0 tehÃ¡t, a fÅÃ¡tlÃ³ minden elemÃ©bÅl kivonunk lambdÃ¡t, Ã©s ennek a mÃ¡trixnak keressÃ¼k a determinÃ¡nsÃ¡t ez egy polinomot fog eredmÃ©nyezni, amiben lambdÃ¡k a vÃ¡ltozÃ³k, Ã©s ennek a polinomnak a gyÃ¶kei lesznek a sajÃ¡tÃ©rtÃ©kek

Ezt a polinomot nevezzÃ¼k a mÃ¡trix *karakterisztikus polinomjÃ¡nak*. ValÃ³s mÃ¡trixnak is lehetnek komplex sajÃ¡rtÃ©rtÃ©kei!

A mÃ¡trix sajÃ¡rtÃ©rtÃ©keinek a halmazÃ¡t a mÃ¡trix *spektrumÃ¡nak* hÃvjuk.

HatvÃ¡nymÃ³dszer

A hatvÃ¡nymÃ³dszer a legnagyobb abszolÃºtÃ©rtÃ©kÅ± sajÃ¡tÃ©rtÃ©k meghatÃ¡rozÃ¡sÃ¡ra szolgÃ¡l. IterÃ¡ciÃ³s mÃ³dszer.

y^k = Ax^k x^(k+1) = y^k/||y^k||

a kiindulÃ¡si x vektor ne legyen a nullvektor, Ã©s nem lehet merÅleges a legnagyobb abszolÃºtÃ©rtÃ©kÃº sajÃ¡tÃ©rtÃ©khez tartozÃ³ sajÃ¡tvektorra.

A k betÅ±k a kitevÅben a k. iterÃ¡ciÃ³t jelentik, nem k. hatvÃ¡nyt.

Inverz hatvÃ¡nymÃ³dszer

Ay=x^k x^(k+1) = y/||y||

Az inverz hatvÃ¡nymÃ³dszer azon a felismerÃ©sen alapul, hogy ha az A mÃ¡trix sajÃ¡tÃ©rtÃ©ke lambda, Ã©s a hozzÃ¡ tartozÃ³ sajÃ¡tvektor x, akkor A^-1 egy sajÃ¡tÃ©rtÃ©ke lambda^-1, Ã©s a hozzÃ¡ tartozÃ³ sajÃ¡tvektor x.

8. ÃrintÅ, szelÅ, Ã©s hÃºr mÃ³dszer, a konjugÃ¡lt gradiens eljÃ¡rÃ¡s. Lagrange interpolÃ¡ciÃ³. Numerikus integrÃ¡lÃ¡s

ÃrintÅ, szelÅ, hÃºrmÃ³dszer, konjugÃ¡lt gradiens eljÃ¡rÃ¡s

Mindegyik egyvÃ¡ltozÃ³s fÃ¼ggvÃ©ny zÃ©rushelyÃ©t keresi, iterÃ¡ciÃ³s mÃ³dszerrel.

ÃrintÅmÃ³dszer

MÃ¡s nÃ©ven Newton-mÃ³dszer

f(x)=0 egyenlet zÃ©rushelyÃ©t keressÃ¼k, ez legyen x*

Ennek egy kÃ¶rnyezetÃ©ben, ha f(x) differenciÃ¡lhatÃ³, vÃ¡lasszunk ebbÅl a kÃ¶rnyezetbÅl egy kezdÅÃ©rtÃ©ket

Az iterÃ¡ciÃ³, amit hasznÃ¡lunk:

x_k+1 = x_k-f(x_k)/f'(x_k)

Magyarul, a kÃ¶vetkezÅ megoldÃ¡st Ãºgy kapjuk, hogy az elÅzÅ megoldÃ¡sbÃ³l kivonjuk a fÃ¼ggvÃ©ny x_k helyen felvett Ã©rtÃ©kÃ©nek Ã©s a fÃ¼ggvÃ©ny derivÃ¡ltjÃ¡nak az x_k pontban felvett Ã©rtÃ©kÃ©nek a hÃ¡nyadosÃ¡t.

Ha az f(x) fÃ¼ggvÃ©ny kÃ©tszer folytonosan differenciÃ¡lhatÃ³ az x* egy kÃ¶rnyezetÃ©ben, akkor van olyan pont, ahonnan indulva a Newton-mÃ³dszer kvadratikusan konvergens sorozatot ad meg, aka gyorsan konvergÃ¡l a megoldÃ¡shoz.

|x-*x_{k+1}*| <= *C*|*x*-x_k|^2

SzelÅmÃ³dszer

Legyen megint x* az f(x)=0 egyenlet egyszeres gyÃ¶ke, Ã©s megint ezt keressÃ¼k iterÃ¡ciÃ³val.

A fÃ¼ggvÃ©ny derivÃ¡ltjÃ¡t nem mindig tudjuk, de a fÃ¼ggvÃ©nyt ki tudjuk Ã©rtÃ©kelni minden helyen. Ekkor f' helyett hasznÃ¡lhatjuk az (f(x_k)-f(x_{k-1}))/(x_k-x_{k-1}) kÃ©pletet.

Ekkor f' helyÃ©re a felsÅ kÃ©pletet behelyettesÃtve megkapjuk a szelÅmÃ³dszer iterÃ¡ciÃ³s kÃ©pletÃ©t:

x_{k+1} = x_k-f(x_k)*(x_k-x_{k-1})/(f(x_k)-f(x_{k-1}))

AzÃ©rt szelÅmÃ³dszer a neve, mert x_{k+1} az az (x_k, f(x_k})) Ã©s (x_{k-1}, f(x_{k-1})) pontokon Ã¡tmenÅ egyenes Ã©s az x tengely metszÃ©spontjÃ¡nak koordinÃ¡tÃ¡ja.

Olyan x0, x1 kezdÅÃ©rtÃ©kekkel szokÃ¡s indÃtani, amelyek kÃ¶zrefogjÃ¡k a gyÃ¶kÃ¶t, amit keresÃ¼nk.

HÃºrmÃ³dszer

A szelÅmÃ³dszer egy vÃ¡ltozata.

FeltesszÃ¼k, hogy a kezdeti x0, x1 pontokban az f(x) fÃ¼ggvÃ©ny ellentÃ©tes elÅjelÅ±, Ã©s f(x_{k+1}) fÃ¼ggvÃ©nyÃ©ben a megelÅzÅ kÃ©t pontbÃ³l azt vÃ¡lasztjuk, amivel ez a tulajdonsÃ¡g fennmarad.

KonjugÃ¡lt gradiens eljÃ¡rÃ¡s

Szimmetrikus, pozitÃv definit mÃ¡trixÃº lineÃ¡ris egyenletrendszerek megoldÃ¡sÃ¡ra alkalmas. Pontos szÃ¡molÃ¡sokkal vÃ©ges sok lÃ©pÃ©sben megtalÃ¡lnÃ¡ a megoldÃ¡st, de a kerekÃtÃ©si hibÃ¡k miatt iterÃ¡ciÃ³s eljÃ¡rÃ¡snak veszik.

q(x) = 1/2x^TAxâx^Tb kvadratikus fÃ¼ggvÃ©ny minimumpontjÃ¡t keressÃ¼k, mert ez ugyanaz, mint az eredeti egyenletrendszerÃ¼nk megoldÃ¡sa, ha lÃ©tezik.

Ãgy keressÃ¼k a kÃ¶vetkezÅ kÃ¶zelÃtÅ megoldÃ¡st, hogy van egy keresÃ©si irÃ¡nyunk, Ã©s egy lÃ©pÃ©skÃ¶zÃ¼nk, Ã©s az aktuÃ¡lis pontbÃ³l lÃ©pÃ¼nk ebbe az irÃ¡nyba ekkora lÃ©pÃ©skÃ¶zzel egyet.

A negatÃv gradiensvektort nevezzÃ¼k reziduÃ¡lis vektornak (erre csÃ¶kken a fÃ¼ggvÃ©nyÃ¼nk). Ez lesz r = b-Ax. A keresÃ©si irÃ¡nyban ott lesz a cÃ©lfÃ¼ggvÃ©ny minimÃ¡lis ahol az Ãºj reziduÃ¡lis vektor merÅleges az elÅzÅ keresÃ©si irÃ¡nyra, szÃ³val tudjuk pontosan, hogy hova kell lÃ©pnÃ¼nk az adott irÃ¡nyban.

TehÃ¡t a konjugÃ¡lt gradiens mÃ³dszer:

- meghatÃ¡rozzuk a lÃ©pÃ©shosszt
- meghatÃ¡rozzuk az Ãºj kÃ¶zelÃtÅ megoldÃ¡st (lÃ©pÃ¼nk egyet az elÅzÅ megoldÃ¡sbÃ³l az adott irÃ¡nyba az Ãºj lÃ©pÃ©shosszal)
- ebbÅl kiszÃ¡moljuk az Ãºj reziduÃ¡lis vektort
- Ã©s az Ãºj keresÃ©si irÃ¡nyt
- Ã©s kezdjÃ¼k elÃ¶lrÅl

A megÃ¡llÃ¡si feltÃ©telÃ¼nk lehet az, hogy az utolsÃ³ nÃ©hÃ¡ny iterÃ¡lt kÃ¶zelÃtÃ©s eltÃ©rÃ©se Ã©s a reziduÃ¡lis vektorok eltÃ©rÃ©se bizonyos kicsi hatÃ¡r alatt maradtak.

Lagrange interpolÃ¡ciÃ³

FÃ¼ggvÃ©nykÃ¶zelÃtÃ©ses mÃ³dszer. Van pÃ¡r alappontunk, Ã©s ezekre szeretnÃ©nk egy polinomot illeszteni. Ezek az alappontok legyenek pÃ¡ronkÃ©nt kÃ¼lÃ¶nbÃ¶zÅek.

Minden pontra felÃrunk egy egyenletet. AhÃ¡ny alappontunk van, max annyiad fokÃº lesz a kapott polinomunk. Az egyenlet Ãºgy fog kinÃ©zni, hogy ismerjÃ¼k az x_i Ã©rtÃ©ket, Ã©s mindenhova behelyettesÃtjÃ¼k Åket, Ã©s ezeknek az x_i^1, x_i^2, stb vÃ¡ltozÃ³knak keressÃ¼k az egyÃ¼tthatÃ³jÃ¡t. Az egyenlet jobb oldalÃ¡n pedig az f(x_i) Ã©rtÃ©kek vannak.

EbbÅl kapunk egy lineÃ¡ris egyenletrendszert, ahol az egyÃ¼tthatÃ³kat keressÃ¼k. Ennek az egyenletrendszernek a mÃ¡trixa egy Vandermonde-mÃ¡trix lesz. EbbÅl kÃ¶vetkezik, hogy pontosan egy polinom lÃ©tezik, ami az adott pontokon Ã¡thalad.

A Lagrange-interpolÃ¡ciÃ³ az interpolÃ¡lÃ³ polinomot a Szumma f(x_i)L_i(x) alakban adja meg. L_i(x)-et Ãºgy kapjuk, hogy egy nagy tÃ¶rtet veszÃ¼nk - a szÃ¡mlÃ¡lÃ³ban Ã¶sszeszorozzuk az Ã¶sszes x-x_j-t, ahol *j nem egyenlÅ i-vel*, tehÃ¡t x-x_i szorzÃ³ kimarad belÅle A nevezÅben pedig x_i-x_j-ket szorzunk Ã¶ssze, mindenhol, ahol j nem egyenlÅ i-vel szintÃ©n (kÃ¼lÃ¶nben nullÃ¡val osztanÃ¡nk).

Numerikus integrÃ¡lÃ¡s

HatÃ¡rozott integrÃ¡lokat akarunk kÃ¶zelÃteni, Ãºgynevezett kvadratÃºra formulÃ¡kkal.

Q_n(f)-fel jelÃ¶ljÃ¼k, Q_n(f)=Szumma i=1-tÅl n-ig w_i * f(x_i)

ÃltalÃ¡ban feltesszÃ¼k, hogy az Ã¶sszes x_i az [a,b] intervallumban van, ugye ebben az intervallumban keressÃ¼k a hatÃ¡rozott integrÃ¡ljÃ¡t f-nek. A w_i szÃ¡mokat pedig sÃºlyoknak hÃvjuk. HomogÃ©n Ã©s additÃv lekÃ©pezÃ©s, azaz kÃ©t fÃ¼ggvÃ©ny Ã¶sszegÃ©nek a hatÃ¡rozott integrÃ¡lja a kÃ©t fÃ¼ggvÃ©ny hatÃ¡rozott integrÃ¡ljÃ¡nak az Ã¶sszege, Ã©s egy fÃ¼ggvÃ©ny szÃ¡mszorosÃ¡nak hatÃ¡rozott integrÃ¡lja a fÃ¼ggvÃ©ny hatÃ¡rozott integrÃ¡ljÃ¡nak szÃ¡mszorosa.

A hatÃ¡rok szerinti additivitÃ¡s fontos tulajdonsÃ¡g, tehÃ¡t pl integrÃ¡l a-tÃ³l b-ig az ugyanaz mint integrÃ¡l a-tÃ³l c-ig plusz integrÃ¡l c-tÅl b-ig, ahol a < c < b

A kvadratÃºra-formula hibÃ¡ja a hatÃ¡rozott integrÃ¡l mÃnusz a kvadratÃºra formula kifejezÃ©ssel definiÃ¡ljuk. Ha ez nulla, akkor pontos a kvadratÃºra formula.

KvadratÃºra formula pontossÃ¡gi rendje az *r* termÃ©szetes szÃ¡m, ha az pontos az 1, x, x^2, x^3, ..., x^r hatvÃ¡nyfÃ¼ggvÃ©nyekre, de nem pontos x^{r+1}-re. A rend meghatÃ¡rozÃ¡sa ekvivalens egy egyenletrendszer megoldÃ¡sÃ¡val. Ha az alappontokat (tehÃ¡t x1, x2, stb) ismeretlennek tekintjÃ¼k, akkor ez egy r+1 egyenletbÅl Ã¡llÃ³ egyenletrendszer (mert elmegyÃ¼nk x^r-ig, plusz az x^0, azaz 1), amiben 2n vÃ¡ltozÃ³ van (n sÃºly Ã©s n darab x).

Az n alappontos kvadratÃºra formula rendje legfeljebb 2n-1 lehet.

10. NormÃ¡lformÃ¡k a predikÃ¡tumkalkulusban. EgyesÃtÃ©si algoritmus. KÃ¶vetkeztetÅ mÃ³dszerek: Alap rezolÃºciÃ³, elsÅrendÅ± rezolÃºciÃ³

NormÃ¡lformÃ¡k predikÃ¡tumkalkulusban

Prenex alak

- eliminÃ¡ljuk a nyilakat
- kiigazÃtjuk a formulÃ¡t (vÃ¡ltozÃ³kat Ã¡tnevezzÃ¼k, ha van vÃ¡ltozÃ³nÃ©v-Ã¼tkÃ¶zÃ©s)
- az Ã¶sszes kvantort kihozzuk a formula elejÃ©re, ha pÃ¡ratlan negÃ¡lÃ¡s scope-jÃ¡ban volt, akkor fordul, ha pÃ¡ros, nem

Skolem alak

- prenex alak
- a lÃ©tezik kvantorhoz tartozÃ³ vÃ¡ltozÃ³kat lecserÃ©ljÃ¼k Ãºj fÃ¼ggvÃ©nyekre, amik az elÅtte Ã¡llÃ³ bÃ¡rmely-kvantÃ¡lt vÃ¡ltozÃ³ktÃ³l fÃ¼ggnek

ZÃ¡rt Skolem alak

- Skolem alak
- a szabad vÃ¡ltozÃ³kat lecserÃ©ljÃ¼k konstansokra, pl minden x helyÃ©re cx-et Ãrunk

EgyesÃtÃ©si algoritmus

Ha F egy formula, akkor F[x/t] azt jelenti, hogy F-ben x Ã¶sszes elÅfordulÃ¡sÃ¡t helyettesÃtjÃ¼k t-vel.

Ha x1, x2, ..., xn vÃ¡ltozÃ³k, Ã©s t1, ..., tn termek, akkor az [x1/t1], ..., [xn/tn] helyettesÃtÃ©s azt jelenti, hogy elÅszÃ¶r x1 helyÃ©re Ãrunk t1-et, aztÃ¡n az eredmÃ©nyben x2 helyÃ©re t2-t, stb.

FormulÃ¡k halmazaira, pl klÃ³zokra is Ã©rtelmezhetjÃ¼k ezt.

KlÃ³z vÃ©gzett helyettesÃtÃ©snÃ©l [x/t] azt jelenti, hogy minden klÃ³zra elvÃ©gezzÃ¼k az x helyÃ©re t helyettesÃtÃ©st, Ã©s az eredmÃ©nyeket visszapakoljuk egy halmazba. Ha C={l1, l2, ..., ln} literÃ¡lok halmaza, akkor s a c egyesÃtÅje, ha l1*s = ... = ln*s. C-re akkor mondjuk, hogy egyesÃthetÅ, ha van egyesÃtÅje.

Az s helyettesÃtÃ©s Ã¡ltalÃ¡nosabb az s' helyettesÃtÃ©snÃ©l, ha van olyan s'' helyettesÃtÃ©s, hogy s*s'' = s'.

EgyesÃtÃ©si algoritmus:

- input: C klÃ³z
- output: C legÃ¡ltalÃ¡nosabb helyettesÃtÅje, ha egyesÃthetÅ, kÃ¼lÃ¶nben azzal tÃ©r vissza, hogy nem egyesÃthetÅ
- veszÃ¼nk kÃ©t literÃ¡lt, Ã©s keressÃ¼k az elsÅ eltÃ©rÃ©st
- ha az egyik helyen egy x vÃ¡ltozÃ³ Ã¡ll, a mÃ¡sikon egy t term, amiben nincs x, akkor x/t Ã©s vissza az elÅzÅ pontra
- kÃ¼lÃ¶nben return nem egyesÃthetÅ

Nem egyesÃthetÅ pl

- ha f(x) Ã©s c a kÃ¼lÃ¶nbsÃ©g a kÃ©t literÃ¡l azonos pontjÃ¡n
- ha x Ã©s f(x) a kÃ¼lÃ¶nbsÃ©g
- ha g(x) Ã©s f(x) a kÃ¼lÃ¶nbsÃ©g

AlaprezolÃºciÃ³

- input: elsÅrendÅ± formulÃ¡k egy szigma halmaza
- output: kielÃ©gÃthetetlen vÃ©ges sok lÃ©pÃ©sben, vagy kielÃ©gÃthetÅ vÃ©ges sokban vagy vÃ©gtelen ciklus
- szigma elemeit zÃ¡rt skolem alakra hozzuk, a formula belsejÃ©t pedig CNF-re, ez legyen szigma'
- ekkor E(szigma') a klÃ³zok alappÃ©ldÃ¡nyainak a halmaza
- E(szigma')-n futtatjuk az ÃtÃ©letkalkulusbeli rezolÃºciÃ³s algoritmust
- E(szigma') Ã¡ltalÃ¡ban vÃ©gtelen
- vegyÃ¼k fel E(szigma') egy elemÃ©t, Ã©s rezolvÃ¡ljunk vele, amÃg lehet
- ha kijÃ¶n az Ã¼res klÃ³z, akkor jÃ³k vagyunk, ha nem, generÃ¡lunk tovÃ¡bb

ElsÅrendÅ± rezolÃºciÃ³

- input: elsÅrendÅ± formulÃ¡k egy szigma halmaza
- output: kielÃ©gÃthetetlen-e?
- szigma zÃ¡rt skolemre, mag cnfre, szigma'
- szigma' elemeit kÃ¶zvetlenÃ¼l felvehetjÃ¼k a listÃ¡ra
- ha kijÃ¶n az Ã¼res klÃ³z, kielÃ©gÃthetetlen
- ha nem tudunk tÃ¶bb klÃ³zt levezetni, kielÃ©gÃthetÅ

RezolvenskÃ©pzÃ©s:

- C1 Ã©s C2 klÃ³zokat akarjuk rezolvÃ¡lni
- Ã¡tnevezzÃ¼k a vÃ¡ltozÃ³kat Ãºgy, hogy ne legyen kÃ¶zÃ¶s vÃ¡ltozÃ³ C1-ben Ã©s C2-ben
- kivÃ¡lasztunk C1-bÅl Ã©s C2-bÅl is literÃ¡lokat, az egyikbÅl pozitÃvokat, a mÃ¡sikbÃ³l negatÃvokat
- ezeket pozitÃvan belepakoljuk egy C halmazba
- ha C egyesÃthetÅ egy s helyettesÃtÃ©ssel, akkor vehetjÃ¼k a rezolvensÃ©t C1-nek Ã©s C2-nek
- elmentjÃ¼k s-t
- vesszÃ¼k C1-bÅl Ã©s C2-bÅl a maradÃ©k literÃ¡lokat, Ã©s berakjuk egy halmazba
- ezen a halmazon elvÃ©gezzÃ¼k az s helyettesÃtÃ©st, ez lesz a rezolvens

9. NormÃ¡lformÃ¡k az ÃtÃ©letkalkulusban, teljes rendszerek. KÃ¶vetkeztetÅ mÃ³dszerek: Hilbert-kalkulus Ã©s rezolÃºciÃ³

NormÃ¡lformÃ¡k az ÃtÃ©letkalkulusban

DiszjunktÃv normÃ¡lforma

A formula olyan alakja:

- a vÃ¡ltozÃ³k pozitÃvan vagy negatÃvan szerepelhetnek benne
- a zÃ¡rÃ³jelekben lÃ©vÅ pozitÃv vagy negatÃv vÃ¡ltozÃ³k kÃ¶zÃ¶tt Ã©selÃ©s van
- a zÃ¡rÃ³jelek kÃ¶zÃ¶tt vagyolÃ¡s van

NyÃlmentes formula

A nyilakat eliminÃ¡ljuk a formulÃ¡bÃ³l a kÃ¶vetkezÅ szabÃ¡lyok alkalmazÃ¡sÃ¡val:

- F -> G == -F || G

- F <-> G == (F -> G) && (G -> F) == (-F || G) && (-G || F)

NNF

A negÃ¡ciÃ³kat bevisszÃ¼k teljesen a vÃ¡ltozÃ³k elÃ©, hogy semmilyen zÃ¡rÃ³jeles kifejezÃ©s elÅtt ne szerepeljen negÃ¡ciÃ³. Ez a formula mÃ¡r nyÃlmentes is. Ehhez a De Morgan szabÃ¡lyokat alkalmazzuk:

- -(F || G) == -F && -G
- -(F && G) == -F || -G

CNF

A CNF alakban klÃ³zok vannak, Ã©s a klÃ³zok vannak Ã¶sszeÃ©selve egymÃ¡ssal. Egy klÃ³zban vÃ¡ltozÃ³k vannak, negatÃvan vagy pozitÃvan, Ã©s ezek kÃ¶zÃ¶tt vagyolÃ¡s van. Ãgy kapjuk, hogy egy mÃ¡r NNF-ben lÃ©vÅ formulÃ¡ban alkalmazzuk a disztribÃºciÃ³s szabÃ¡lyt:

- (F && G) || H == (F || H) && (G || H)
- (F && G) || (H && I) == (F || H) && (F || I) && (G || H) && (G || I)

Teljes rendszerek

Logikai mÅ±veletek egy rendszerÃ©t akkor nevezzÃ¼k teljesnek, ha egy, mÃ¡r korÃ¡bban teljesnek ÃtÃ©lt rendszer minden mÅ±veletÃ©t ki tudjuk fejezni ezen mÅ±veletekkel. A {-, &&, ||} rendszer teljes, mert minden formulÃ¡t CNF alakra tudunk hozni. Ezek alapjÃ¡n teljes mÃ©g:

- {-, &&}
-
    - A negÃ¡ciÃ³ okÃ©s, az Ã©selÃ©s okÃ©s, a vagyolÃ¡st ki tudjuk fejezni:
-
    -
        - p || q == -(-p && -q)
- {-, ||}
-
    - A negÃ¡ciÃ³ okÃ©s, a vagyolÃ¡s okÃ©s, az Ã©selÃ©st ki tudjuk fejezni:
-
    -
        - p && q == -(-p || -q)

A {-, ->} rendszer is teljes, mert tudjuk, hogy a {-, ||} rendszer teljes, Ã©s ki tudjuk fejezni a mÅ±veleteit:

- negÃ¡ciÃ³ okÃ©s, vagyolÃ¡s:
-
    - p || q == (-p) -> q

A {->, lenyÃl} rendszer is teljes, mert tudjuk, hogy a {-, ->} rendszer teljes, Ã©s ki tudjuk fejezni a mÅ±veleteit:

- nyÃl okÃ©s
- -p == p -> lenyÃl

Ezt a rendszert nevezzÃ¼k Hilbert rendszerÃ©nek.

RezolÃºciÃ³

A rezolÃºciÃ³nÃ¡l a formulÃ¡ink CNF alakban vannak. A rezolÃºciÃ³val logikai kÃ¶vetkezmÃ©nyeket tudunk bebizonyÃtani, pl. hogy egy formulahalmaznak logikai kÃ¶vetkezmÃ©nye egy formula.

AlapbÃ³l a logikai kÃ¶vetkezmÃ©ny azt jelenti, hogy azoknak az Ã©rtÃ©kadÃ¡sonak a halmaza, amelyek kielÃ©gÃtik a jobboldali formulÃ¡(ka)t, rÃ©szhalmaza a jobboldali formulÃ¡kat kielÃ©gÃtÅ Ã©rtÃ©kadÃ¡sok halmazÃ¡nak. Ezzel az a baj, hogy az Ã¶sszes ilyen Ã©rtÃ©kadÃ¡st megtalÃ¡lni nagyon hosszadalmas.

A rezolÃºciÃ³s algoritmus inputja klÃ³zoknak egy halmaza, Ã©s outputja egy igen vagy egy nem, attÃ³l fÃ¼ggÅen, hogy kielÃ©gÃthetÅ vagy kielÃ©gÃthetetlen ez a klÃ³zhalmaz. A baloldali formulÃ¡k kÃ¶zÃ© felvesszÃ¼k elÅszÃ¶r a jobboldali formula negÃ¡ltjÃ¡t, hiszen ha az Ãgy kapott Ãºj formulahalmaz kielÃ©gÃthetetlen (azaz Mod(Szigma) Ã¼reshalmaz), akkor az eredeti logikai kÃ¶vetkezmÃ©ny fennÃ¡ll.

EzutÃ¡n listÃ¡t vezetÃ¼nk a klÃ³zokrÃ³l. Egy klÃ³z felkerÃ¼lhet a listÃ¡ra, ha

- eleme a SzigmÃ¡nak
- kÃ©t, korÃ¡bban mÃ¡r a listÃ¡n szereplÅ klÃ³z rezolvense

KÃ©t klÃ³znak akkor vehetjÃ¼k a rezolvensÃ©t, ha a mindkettÅben szerepel ugyanaz a vÃ¡ltozÃ³, de az egyikben negatÃvan, a mÃ¡sikban pedig pozitÃvan. Ekkor a rezolvens egy olyan klÃ³z lesz, ahol ez a vÃ¡ltozÃ³ mÃ¡r nem fog szerepelni, hanem csak a kÃ©t klÃ³zban maradt Ã¶sszes tÃ¶bbi vÃ¡ltozÃ³.

Ha a listÃ¡ra valamelyik lÃ©pÃ©sben rÃ¡kerÃ¼l az Ã¼resklÃ³z, az azt jelenti, hogy Szigma kielÃ©gÃthetetlen, vagyis az eredeti logikai kÃ¶vetkezmÃ©ny fennÃ¡ll. Ha sehogy sem tudjuk levezetni az Ã¼resklÃ³zt, az azt jelenti, hogy a Szigma kielÃ©gÃthetÅ, Ã©s az eredeti logikai kÃ¶vetkezmÃ©ny nem Ã¡ll fenn.

Hilbert-kalkulus

A Hilbert-kalkulusban Hilbert rendszerÃ©t hasznÃ¡ljuk. Az ilyen alakÃº formulÃ¡kra is tudunk kÃ¶vetkeztetÅ rendszert Ã©pÃteni. A tovÃ¡bbiakban a formulÃ¡ink mind Hilbert rendszerÃ©bÅl szÃ¡rmaznak.

A kÃ¶vetkeztetÅ rendszerÃ¼nkben az input szintÃ©n egy formulahalmaz, illetve egy formula, amirÅl be akarjuk lÃ¡tni, hogy logikai kÃ¶vetkezmÃ©nye a formulahalmazunknak.

Ekkor a formulÃ¡krÃ³l szintÃ©n listÃ¡t vezetÃ¼nk, ahol a listÃ¡ra felkerÃ¼lhet egy formula, ha:

- benne van a SzigmÃ¡ban
- axiÃ³mapÃ©ldÃ¡ny
- modus ponense kÃ©t, korÃ¡bban a listÃ¡n szereplÅ formulÃ¡nak

HÃ¡romfÃ©le axiÃ³mÃ¡nk van: Ax1: (F -> (G -> H)) -> ((F -> G) -> (F -> H)) Ax2: F -> (G -> F) Ax3: ((F -> lenyÃl) -> lenyÃl) -> F

KÃ©t formulÃ¡nak vehetjÃ¼k a modus ponensÃ©t, ha az egyik formula F, a mÃ¡sik pedig F -> G alakÃº. Ekkor a modus ponens pontosan G lesz.

Ezekkel a szabÃ¡lyokkal ha a listÃ¡ra kerÃ¼l a logikai kÃ¶vetkezmÃ©ny jobb oldalÃ¡n szereplÅ formula, akkor igazoltuk a logikai kÃ¶vetkezmÃ©nyt.

Ãrdemes mÃ©g az algoritmus elÅtt a dedukciÃ³ mÅ±velettel kezdeni. Ha a jobb oldali formula F -> G alakÃº, akkor F-et Ã¡tvehetjÃ¼k a SzigmÃ¡ba, Ã©s ezt mindaddig ismÃ©telhetjÃ¼k, amÃg a jobb oldal ilyen alakÃº.# 11. KeresÃ©si feladat: feladatreprezentÃ¡ciÃ³, vak keresÃ©s, informÃ¡lt keresÃ©s, heurisztikÃ¡k. KÃ©tszemÃ©lyes zÃ©rÃ³ Ã¶sszegÅ± jÃ¡tÃ©kok: minimax, alfa-bÃ©ta eljÃ¡rÃ¡s. KorlÃ¡tozÃ¡s kielÃ©gÃtÃ©si feladat

KeresÃ©si feladat: feladatreprezentÃ¡ciÃ³, vak keresÃ©s, informÃ¡lt keresÃ©s, heurisztikÃ¡k

FeladatreprezentÃ¡ciÃ³

TekintsÃ¼nk egy diszkrÃ©t, statikus, determinisztikus Ã©s teljesen megfigyelhetÅ feladatkÃ¶rnyezetet. TegyÃ¼k fel, hogy a vilÃ¡g tÃ¶kÃ©letesen modellezhetÅ a kÃ¶vetkezÅkkel:

- lehetsÃ©ges Ã¡llapotok halmaza
- egy kezdÅÃ¡llapot
- lehetsÃ©ges cselekvÃ©sek halmaza (Ã¡llapotÃ¡tmenet fÃ¼ggvÃ©ny, minden Ã¡llapothoz hozzÃ¡rendelÃ¼nk egy (cselekvÃ©s, Ã¡llapot) rendezett pÃ¡rokbÃ³l Ã¡llÃ³ halmazt, tehÃ¡t egy Ã¡llapotban milyen cselekvÃ©sek hatÃ¡sÃ¡ra milyen Ã¡llapotba juthat az Ã¡gensÃ¼nk)
- Ã¡llapotÃ¡tmenet kÃ¶ltsÃ©gfÃ¼ggvÃ©nye, minden lehetsÃ©ges Ã¡llapot-cselekvÃ©s-Ã¡llapot hÃ¡rmashoz hozzÃ¡rendelÃ¼nk egy kÃ¶ltsÃ©get, azaz egy Ã¡llapotbÃ³l egy (mÃ¡sik) Ã¡llapotba jutÃ¡snak mekkora a kÃ¶ltsÃ©ge
- cÃ©lÃ¡llapotok halmaza, tehÃ¡t hova szeretnÃ©nk, hogy eljusson az Ã¡gensÃ¼nk

Ez egy sÃºlyozott grÃ¡fot definiÃ¡l, ez a grÃ¡f az Ã¡llapottÃ©r

FeltesszÃ¼k tovÃ¡bbÃ¡, hogy az Ã¡llapotok szÃ¡ma vÃ©ges, vagy megszÃ¡mlÃ¡lhatÃ³. Ãton Ã¡llapotok cselekvÃ©sekkel Ã¶sszekÃ¶tÃ¶tt sorozatÃ¡t Ã©rtjÃ¼k, ennek van egy Ã¶sszkÃ¶ltsÃ©ge is.

Vak (informÃ¡latlan) keresÃ©s

FakeresÃ©s

Adott kezdÅÃ¡llapotbÃ³l talÃ¡ljunk minimÃ¡lis kÃ¶ltsÃ©gÅ± utat egy cÃ©lÃ¡llapotba. Az Ã¡llapottÃ©r nem mindig adott explicit mÃ³don, Ã©s vÃ©gtelen is lehet.

Ãtlet: keresÅfa Ã©pÃtÃ©se, a kezdÅÃ¡llapotbÃ³l nÃ¶vesszÃ¼nk fÃ¡t a szomszÃ©dos Ã¡llapotok hozzÃ¡vÃ©telÃ©vel, amÃg cÃ©lÃ¡llapotot nem talÃ¡lunk. A keresÅfa NEM azonos a feladat Ã¡llapotterÃ©vel, pl ha van kÃ©t csÃºcs kÃ¶zÃ¶tt oda-vissza Ã©l.

fakeresÃ©s 1 perem = { Ãºj csÃºcs(kezdÅÃ¡llapot) } 2 while perem.nemÃ¼res() 3 csÃºcs = perem.elsÅkivesz() 4 if csÃºcs.cÃ©lÃ¡llapot() return csÃºcs 5 perem.beszÃºr(csÃºcs.kiterjeszt()) 6 return failure

A csÃºcs.kiterjeszt() lÃ©trehozza a csÃºcsbÃ³l elÃ©rhetÅ Ã¶sszes Ã¡llapotbÃ³l a keresÅfa csÃºcsot. A perem egy prioritÃ¡si sor, ettÅl fÃ¼gg a bejÃ¡rÃ¡si stratÃ©gia.

A hatÃ©konysÃ¡got nÃ¶velhetjÃ¼k, ha Ãºgy szÃºrunk be csÃºcsokat a perembe, hogy abban az esetben, ha a peremben talÃ¡lhatÃ³ mÃ¡r ugyanazzal az Ã¡llapottal egy mÃ¡sik csÃºcs, akkor ha az Ãºj csÃºcs kÃ¶ltsÃ©ge kisebb, lecserÃ©ljÃ¼k a rÃ©gi csÃºcsot az Ãºjra, kÃ¼lÃ¶nben nem tesszÃ¼k bele az Ãºjat.

SzÃ©lessÃ©gi keresÃ©s

FakeresÃ©s, ahol a perem egy FIFO perem.

- Teljes, minden, vÃ©ges szÃ¡mÃº Ã¡llapot Ã©rintÃ©sÃ©vel elÃ©rhetÅ Ã¡llapotot vÃ©ges idÅben elÃ©r
- ÃltalÃ¡ban nem optimÃ¡lis, de pl akkor igen, ha a kÃ¶ltsÃ©g a mÃ©lysÃ©g nem csÃ¶kkenÅ fÃ¼ggvÃ©nye
- idÅigÃ©ny = tÃ¡rigÃ©ny O(b^{d+1})
    - b: szomszÃ©dok maximÃ¡lis szÃ¡ma
    - d: a legkisebb mÃ©lysÃ©gÅ± cÃ©lÃ¡llapot mÃ©lysÃ©ge

MÃ©lysÃ©gi keresÃ©s

FakeresÃ©s, LIFO perem

- Teljes, ha a keresÃ©si fa vÃ©ges mÃ©lysÃ©gÅ±
- Nem optimÃ¡lis
- IdÅigÃ©ny: legrosszabb esetben O(b^m) (nagyon rossz, lehet vÃ©gtelen), tÃ¡rigÃ©ny legrosszabb esetben O(bm) (ez egÃ©sz bÃztatÃ³)

IteratÃvan mÃ©lyÃ¼lÅ keresÃ©s

MÃ©lysÃ©gi keresÃ©sek sorozata 1, 2, 3 stb mÃ©ylsÃ©gre korlÃ¡tozva, amÃg cÃ©lÃ¡llapotot nem talÃ¡lunk.

- TeljessÃ©g Ã©s optimalitÃ¡s a szÃ©lessÃ©givel egyezik meg
- idÅigÃ©ny = O(b^d) (akÃ¡r jobb is lehet, mint a szÃ©lessÃ©gi), tÃ¡rigÃ©ny = O(bd) (jobb, mint a mÃ©lysÃ©gi)

Ez a legjobb informÃ¡latlan keresÅ.

Egyenletes kÃ¶ltsÃ©gÅ± keresÃ©s

A peremben a rendezÃ©s kÃ¶ltsÃ©g alapÃº, mindig elÅszÃ¶r a legkisebb ÃºtkÃ¶ltsÃ©gÅ± csÃºcsot terjesztjÃ¼k ki.

- Teljes Ã©s optimÃ¡lis, ha minden Ã©l kÃ¶ltsÃ©ge nagyobb mint nulla
- IdÅ Ã©s tÃ¡rigÃ©ny nagyban fÃ¼gg a kÃ¶ltsÃ©gfÃ¼ggvÃ©nytÅl

GrÃ¡fkeresÃ©s

Ha nem fa az Ã¡llapottÃ©r!

FakeresÃ©s, de a perem mellett mÃ©g tÃ¡rolunk egy Ãºn. zÃ¡rt halmazt is. A zÃ¡rt halmazba azok a csÃºcsok kerÃ¼lnek, amiket mÃ¡r kiterjesztettÃ¼nk. A perembe helyezÃ©s elÅtt minden csÃºcsra megnÃ©zzÃ¼k, hogy mÃ¡r a zÃ¡rt halmazban van-e. Ha igen, nem tesszÃ¼k a perembe. MÃ¡srÃ©szt minden perembÅl kivett csÃºcsot a zÃ¡rt halmazba teszÃ¼nk. Ãgy minden Ã¡llapothoz a legelsÅ megtalÃ¡lt Ãºt lesz tÃ¡rolva.

InformÃ¡lt keresÃ©s, heurisztikÃ¡k

Itt mÃ¡r tudjuk, hogy "hova megyÃ¼nk".

Heurisztika: minden Ã¡llapotbÃ³l megbecsÃ¼li, hogy mekkora az optimÃ¡lis Ãºt kÃ¶ltsÃ©ge az adott Ã¡llapotbÃ³l egy cÃ©lÃ¡llapotba: tehÃ¡t Ã©rtelmesebben tudunk kÃ¶vetkezÅ szomszÃ©dot vÃ¡lasztani. Pl. lÃ©gvonalbeli tÃ¡volsÃ¡g a cÃ©lig a tÃ©rkÃ©pen egy Ãºtvonal-tervezÃ©si problÃ©mÃ¡hoz jÃ³ heurisztika.

h(n): optimÃ¡lis kÃ¶ltsÃ©g kÃ¶zelÃtÃ©se a legkÃ¶zelebbi cÃ©lÃ¡llapotba g(n): tÃ©nyleges kÃ¶ltsÃ©g a kezdÅÃ¡llapotbÃ³l a jelenlegi Ã¡llapotba

MohÃ³

FakeresÃ©s, peremben a rendezÃ©st h() alapjÃ¡n csinÃ¡ljuk, mindig a legkisebb Ã©rtÃ©kÅ± csÃºcsot vesszÃ¼k ki.

- Teljes, de csak ha a keresÃ©si fa vÃ©ges mÃ©lysÃ©gÅ±
- Nem optimÃ¡lis
- idÅigÃ©ny, tÃ¡rigÃ©ny O(b^m)

A*

A peremben a rendezÃ©st f()=h()+g() alapjÃ¡n vÃ©gezzÃ¼k, a legkisebb csÃºcsot vesszÃ¼k ki. f() a teljes Ãºt kÃ¶ltsÃ©gÃ©t becsÃ¼li a kezdÅÃ¡llapotbÃ³l a vÃ©gÃ¡llapotba. Ha h = 0, Ã©s grÃ¡fkeresÃ©st alkalmazunk, akkor a Dijkstra-t kapjuk.

Egy h heurisztika elfogadhatÃ³, ha nem ad nagyobb Ã©rtÃ©ket, mint a tÃ©nyleges optimÃ¡lis Ã©rtÃ©k. FakeresÃ©st feltÃ©telezve, ha h elfogadhatÃ³ Ã©s a keresÃ©si fa vÃ©ges, akkor A* optimÃ¡lis.

Egy h heurisztika konzisztens, ha h(n) <= mint a valÃ³di kÃ¶ltsÃ©g n egyik bÃ¡rmely, plusz a szomszÃ©d heurisztikÃ¡ja. GrÃ¡fkeresÃ©st feltÃ©telezve, ha h konzisztens Ã©s az Ã¡llapottÃ©r vÃ©ges, akkor A* optimÃ¡lis.

Az A* optimÃ¡lisan hatÃ©kony, de a tÃ¡rigÃ©nye Ã¡ltalÃ¡ban exponenciÃ¡lis, Ã©s nagyon nagyban fÃ¼gg h-tÃ³l. Az idÅigÃ©ny szintÃ©n nagyon nagyban fÃ¼gg h-tÃ³l.

HeurisztikÃ¡k

A relaxÃ¡lt problÃ©ma optimÃ¡lis megoldÃ¡sa pl jÃ³ heurisztika lehet.

RelaxÃ¡lt problÃ©ma: elhagyunk feltÃ©teleket az eredeti problÃ©mÃ¡bÃ³l. KombinÃ¡lhatunk tÃ¶bb heurisztikÃ¡t is. KÃ©szÃthetÃ¼nk mintaadatbÃ¡zisokat, ahol rÃ©szproblÃ©mÃ¡k egzakt kÃ¶ltsÃ©gÃ©t tÃ¡roljuk.

KÃ©tszemÃ©lyes zerÃ³ Ã¶sszegÅ± jÃ¡tÃ©kok: miminax, alfa-bÃ©ta eljÃ¡rÃ¡s

KÃ©tszemÃ©lyes, lÃ©pÃ©svÃ¡ltÃ¡sos, determinisztikus, zÃ©rÃ³ Ã¶sszegÅ± jÃ¡tÃ©k

- lehetsÃ©ges Ã¡llapotok halmaza
- egy kezdÅÃ¡llapot
- lehetsÃ©ges cselekvÃ©sek halmaza, Ã©s egy Ã¡llapotÃ¡tmenet fÃ¼ggvÃ©ny
- cÃ©lÃ¡llapotok
- hasznossÃ¡gfÃ¼ggvÃ©ny

KÃ©t Ã¡gens van, felvÃ¡ltva lÃ©pnek. Az egyik maximalizÃ¡lni akarja a hasznossÃ¡gfÃ¼ggvÃ©nyt (MAX jÃ¡tÃ©kos), a mÃ¡sik minimalizÃ¡lni (MIN jÃ¡tÃ©kos). KonvenciÃ³ szerint MAX kezd. Az elsÅ cÃ©lÃ¡llapot elÃ©rÃ©sekor a jÃ¡tÃ©knak definÃciÃ³ szerint vÃ©ge.

ZÃ©rÃ³ Ã¶sszegÅ± jÃ¡tÃ©k: A MIN jÃ¡tÃ©kos minimalizÃ¡lja a hasznossÃ¡got, ami ugyanaz, mint maximalizÃ¡lni a negatÃv hasznossÃ¡got. Ez a negamax formalizmus. Itt a kÃ©t jÃ¡tÃ©kos nyeresÃ©gÃ©nek az Ã¶sszege a vÃ©gÃ¡llapotban mindig nulla, innen a zÃ©rÃ³ Ã¶sszegÅ± elnevezÃ©s.

Minimax algoritmus, alfa-bÃ©ta vÃ¡gÃ¡s

MindkÃ©t jÃ¡tÃ©kos ismeri a teljes jÃ¡tÃ©kgrÃ¡fot, bÃ¡rmilyen komplex szÃ¡mÃtÃ¡st kÃ©pes elvÃ©gezni Ã©s nem hibÃ¡zik (tÃ¶kÃ©letes racionalitÃ¡s). A minimax algoritmus alapjÃ¡n lehet megvalÃ³sÃtani a legjobb stratÃ©giÃ¡t tÃ¶kÃ©letes racionalitÃ¡s esetÃ©n.

Minimax:

maxÃ‰rtÃ©k(n) 1 if vÃ©gÃ¡llapot(n) return hasznossÃ¡g(n) 2 max = -vÃ©gtelen 3 for a in n szomszÃ©dai 4 max = max(max, minÃ‰rtÃ©k(a)) 5 return max

minÃ‰rtÃ©k(n) 1 if vÃ©gÃ¡llapot(n) return hasznossÃ¡g(n) 2 min = +vÃ©gtelen 3 for a in n szomszÃ©dai 4 min = min(min, maxÃ‰rtÃ©k(a)) 5 return min

Ha n vÃ©gÃ¡llapot, visszaadja a hasznossÃ¡gÃ¡t. KÃ¼lÃ¶nben a max-nÃ¡l n szomszÃ©daira kiszÃ¡molja a maximÃ¡lis Ã©rtÃ©ket, ami vagy az aktuÃ¡lis maximum, vagy nÃ©zi, hogy a mÃ¡sik jÃ¡tÃ©kos mit lÃ©pne. Csak elmÃ©leti jelentÅsÃ©gÅ±, a minimax algoritmus nem skÃ¡lÃ¡zÃ³dik. Az Ã¶sszes lehetsÃ©ges Ã¡llapot kiszÃ¡molÃ¡sa rettentÅ sok idÅ lenne pl sakknÃ¡l.

Alfa-bÃ©ta vÃ¡gÃ¡s

Ha tudjuk, hogy pl MAX-nak mÃ¡r van egy olyan stratÃ©giÃ¡ja, ahol biztosan egy 10 Ã©rtÃ©kÅ± hasznossÃ¡got el tud Ã©rni az adott csÃºcsban, akkor a csÃºcs tovÃ¡bbi kiÃ©rtÃ©kelÃ©sekor nem kell vizsgÃ¡lni olyan csÃºcsokat, ahol MIN ki tud kÃ©nyszerÃteni <= 10 hasznossÃ¡got, mert ennÃ©l mÃ¡r MAX-nak van jobb stratÃ©giÃ¡ja minÃrtÃ©k Ã©s maxÃrtÃ©k hÃvÃ¡sakor Ã¡tadjuk az alfa Ã©s bÃ©ta paramÃ©tereket is a fÃ¼ggvÃ©nynek.

Alfa jeletÃ©se: MAXnak mÃ¡r felfedeztÃ¼nk egy olyan stratÃ©giÃ¡t, amely alfa hasznossÃ¡got biztosÃt, ha ennÃ©l kisebbet talÃ¡lnÃ¡nk, azt nem vizsgÃ¡ljuk BÃ©ta jelentÃ©se: MINnek mÃ¡r felfedeztÃ¼nk egy olyan stratÃ©giÃ¡t, amely bÃ©ta hasznossÃ¡got biztosÃt, ha ennÃ©l nagyobbat talÃ¡lnÃ¡nk, azt nem vizsgÃ¡ljuk

A gyakorlatban a minimax Ã©s az alfa-bÃ©ta vÃ¡gÃ¡sos algoritmusokat is csak meghatÃ¡rozott mÃ©lysÃ©gig vizsgÃ¡ljuk, illetvve heurisztikÃ¡kat is alkalmazhatunk. A csÃºcsok bejÃ¡rÃ¡si sorrendje is nagyon fontos, mert pl alfa bÃ©ta vÃ¡gÃ¡snÃ¡l egy jÃ³ rendezÃ©s mellett nagyon sok csÃºcsot vÃ¡ghatunk le.

KorlÃ¡tozÃ¡s kielÃ©gÃtÃ©si feladat

A feladat az Ã¡llapottÃ©rrel adott keresÃ©si problÃ©mÃ¡k Ã©s az optimalizÃ¡lÃ¡si problÃ©mÃ¡k jellemzÅit Ã¶tvÃ¶zi. Az Ã¡llapotok Ã©s cÃ©lÃ¡llapotok speciÃ¡lis alakÃºak.

LehetsÃ©ges Ã¡llapotok halmaza: a feladat Ã¡llapotai az n db vÃ¡ltozÃ³ lehetsÃ©ges kombinÃ¡ciÃ³i CÃ©lÃ¡llapotok: a megengedett Ã¡llapotok, adottak kÃ¼lÃ¶nbÃ¶zÅ korlÃ¡tozÃ¡sok, Ã©s azok az Ã¡llapotok a cÃ©lÃ¡llapotok, amik minden korlÃ¡tozÃ¡st kielÃ©gÃtenek.

Az Ãºt a megoldÃ¡sig lÃ©nyegtelen, Ã©s gyakran cÃ©lfÃ¼ggvÃ©ny is Ã©rtelmezve van az Ã¡llapotok felett, ilyenkor egy optimÃ¡lis cÃ©lÃ¡llapot megtalÃ¡lÃ¡sa a cÃ©l.# 12. Teljes egyÃ¼ttes eloszlÃ¡s tÃ¶mÃ¶r reprezentÃ¡ciÃ³ja, Bayes hÃ¡lÃ³k. GÃ©pi tanulÃ¡s: felÃ¼gyelt tanulÃ¡s problÃ©mÃ¡ja, dÃ¶ntÃ©si fÃ¡k, naiv Bayes mÃ³dszer, modellillesztÃ©s, mestersÃ©ges neuronhÃ¡lÃ³k, k-legkÃ¶zelebbi szomszÃ©d mÃ³dszere

Teljes egyÃ¼ttes eloszlÃ¡s tÃ¶mÃ¶r reprezentÃ¡ciÃ³ja, Bayes hÃ¡lÃ³k

Teljes egyÃ¼ttes eloszlÃ¡s

Minden lehetsÃ©ges esemÃ©nyre tudjuk annak a valÃ³szÃnÅ±sÃ©gÃ©t. Pl van 3 logikai tÃpusÃº vÃ©letlen vÃ¡ltozÃ³nk, akkor Ã¶sszesen 2^3=8-fÃ©le eset lehet ezekre. A teljes egyÃ¼ttes eloszlÃ¡snÃ¡l mind a 8 esetnek tudjuk a valÃ³szÃnÅ±sÃ©gÃ©t.

TÃ¶mÃ¶r reprezentÃ¡ciÃ³

A kijelentÃ©sek fÃ¼ggetlensÃ©ge a legfontosabb tulajdonsÃ¡g a teljes egyÃ¼ttes eloszlÃ¡s tÃ¶mÃ¶rÃthetÅsÃ©gÃ©hez. Van fÃ¼ggetlensÃ©g, Ã©s feltÃ©teles fÃ¼ggetlensÃ©g.

FÃ¼ggetlensÃ©g a Ã©s b kijelentÃ©sek fÃ¼ggetlenek, ha P(a Ã©s b) = P(a)*P(b)

A fÃ¼ggetlensÃ©g struktÃºrÃ¡t takar. Ha pl n logikai vÃ¡ltozÃ³nk van, amik kÃ©t fÃ¼ggetlen rÃ©szhalmazra oszthatÃ³k m Ã©s k mÃ©rettel, akkor a 2^n valÃ³szÃnÅ±sÃ©g tÃ¡rolÃ¡sa helyett elÃ©g 2^m+2^k valÃ³szÃnÅ±sÃ©get tÃ¡rolni, ami sokkal kevesebb lehet.

ExtrÃ©m esetben, ha pl. az A1,..., An diszkrÃ©t vÃ¡ltozÃ³k kÃ¶lcsÃ¶nÃ¶sen fÃ¼ggetlenek (tetszÅleges kÃ©t rÃ©szhalmaz fÃ¼ggetlen), akkor csak O(n) Ã©rtÃ©ket kell tÃ¡rolni, mivel ez esetben

P(A1,..., An) = P(A1)...P(An)

FeltÃ©teles fÃ¼ggetlensÃ©g a Ã©s b kijelentÃ©sek feltÃ©telesen fÃ¼ggetlenek c feltevÃ©sÃ©vel, akkor Ã©s csak akkor, ha P(a Ã©s b | c) = P(a|c)*P(b|c). Tipikus eset, ha a Ã©s b kÃ¶zÃ¶s oka c.

Naiv-Bayes szabÃ¡ly Ha A feltevÃ©se mellett B1,...,Bn kÃ¶lcsÃ¶nÃ¶sen fÃ¼ggetlenek, akkor P(B1, . . . , Bn|A) = Produktum^n_i=1 P(Bi|A)

Bayes hÃ¡lÃ³k

A feltÃ©teles fÃ¼ggetlensÃ©g hasznos, mert tÃ¶mÃ¶rÃthetjÃ¼k a teljes egyÃ¼ttes eloszlÃ¡st.

GÃ©pi tanulÃ¡s: felÃ¼gyelt tanulÃ¡s problÃ©mÃ¡ja, dÃ¶ntÃ©si fÃ¡k, naiv Bayes mÃ³dszer, modellillesztÃ©s, mestersÃ©ges neuronhÃ¡lÃ³k, k-legkÃ¶zelebbi szomszÃ©d mÃ³dszere

FelÃ¼gyelt tanulÃ¡s problÃ©mÃ¡ja

Tapasztalati tÃ©nyek felhasznÃ¡lÃ¡sa arra, hogy egy racionÃ¡lis Ã¡gens teljesÃtmÃ©nyÃ©t nÃ¶veljÃ¼k.

FelÃ¼gyelt tanulÃ¡s:

- van az adatok mÃ¶gÃ¶tt valami f: X -> Y fÃ¼ggvÃ©ny, ezt nem ismerjÃ¼k
- adottak tanulÃ³pÃ©ldÃ¡k, amik rendezett pÃ¡rok (x, f(x))

- egy h: X -> Y, fÃ¼ggvÃ©nyt keresÃ¼nk, ami illeszkedik a pÃ©ldÃ¡kra, Ã©s kÃ¶zelÃti f-et
- egy pÃ©ldÃ¡ban az elsÅ elem pl egy email, a mÃ¡sodik pedig egy valamilyen cÃmke, pl spam
- h konzisztens az adatokra, ha h(x)==f(x) minden x tanulÃ³pÃ©ldÃ¡ra
- a h fÃ¼ggvÃ©nyt mindig valami H hipotÃ©zistÃ©rben keressÃ¼k, vagyis valamilyen "alakban"
- a tanulÃ¡s realizÃ¡lhatÃ³, ha van olyan h eleme H, amire h konzisztens
- a gyakorlatban elÃ©g, ha h kÃ¶zel van a pÃ©ldÃ¡khoz, mert a pÃ©ldÃ¡k zajt is tartalmazhatnak, amit kifejezetten kÃ¡ros lenne, ha megtanulna az Ã¡gens (tÃºltanulÃ¡s)
- egy olyan h-t keresÃ¼nk, ami a tanulÃ³pÃ©ldÃ¡kon kÃvÃ¼l is jÃ³l Ã¡ltalÃ¡nosÃt
- nem szabad a tanulÃ³pÃ©ldÃ¡kat bemagolni
- occam borotvÃ¡ja: mindig a legtÃ¶mÃ¶rebb leÃrÃ¡st kell venni
- a priori ismeretek fontosak, a nullÃ¡rÃ³l valÃ³ tanulÃ¡s kb lehetetlen
- szÃ¡mÃtÃ¡si szempontbÃ³l egyszerÅ± reprezentÃ¡ciÃ³ is fontos

DÃ¶ntÃ©si fÃ¡k

InduktÃv (felÃ¼gyelt) tanulÃ¡s konkrÃ©t pÃ©ldÃ¡ja.

FeltesszÃ¼k, hogy X-ben diszkrÃ©t vÃ¡ltozÃ³k egy vektora van, Y-ban pedig szintÃ©n valami diszkrÃ©t vÃ¡ltozÃ³ egy Ã©rtÃ©ke, pl igen-nem

TulajdonkÃ©ppen osztÃ¡lyozÃ¡s, X elemeit kell Y valamelyik osztÃ¡lyÃ¡ba sorolni.

ElÅnye, hogy dÃ¶ntÃ©sei megmagyarÃ¡zhatÃ³k, mert emberileg Ã©rtelmezhetÅ lÃ©pÃ©sekben jutottunk el odÃ¡ig.

KifejezÅereje megegyezik az ÃtÃ©letkalkulusÃ©val.

DÃ¶ntÃ©si fa Ã©pÃtÃ©se

- adottak pozitÃv Ã©s negatÃv pÃ©ldÃ¡k felcÃmkÃ©zve, tipikusan tÃ¶bb szÃ¡z
- vegyÃ¼k a gyÃ¶kÃ©rbe azt a vÃ¡ltozÃ³t, ami a legjobban szeparÃ¡lja a pozitÃv Ã©s negatÃv pÃ©ldÃ¡kat
- ezt folytassuk rekurzÃv mÃ³don
- ha csak pozitÃv vagy negatÃv pÃ©lda van, akkor levÃ©lhez Ã©rtÃ¼nk, felcÃmkÃ©zzÃ¼k ezzel a levelet
- ha Ã¼reshalmaz, akkor a szÃ¼lÅ szerint tÃ¶bbsÃ©gi szavazattal cÃmkÃ©zÃ¼nk
- ha nincs tÃ¶bb vÃ¡ltozÃ³, de vannak negatÃv Ã©s pozitÃv pÃ©ldÃ¡k is, akkor szintÃ©n tÃ¶bbsÃ©gi szavazattal cÃmkÃ©zhetjÃ¼k a levelet

A legjobban szeparÃ¡lÃ³ attribÃºtumot az informÃ¡ciÃ³tartalma, azaz entrÃ³piÃ¡ja segÃtsÃ©gÃ©vel vÃ¡laszthatjuk ki.

Naiv Bayes mÃ³dszer

Statisztikai kÃ¶vetkeztetÃ©si mÃ³dszer, amely adatbÃ¡zisban talÃ¡lhatÃ³ pÃ©ldÃ¡k alapjÃ¡n ismeretlen pÃ©ldÃ¡kat osztÃ¡lyoz.

PÃ©ldÃ¡ul emaileket akarunk spam vagy nem spamkÃ©nt osztÃ¡lyozni. Az emailben lÃ©vÅ szavakra meghatÃ¡rozzuk, hogy milyen valÃ³szÃnÅ±sÃ©ggel fordul elÅ egy normÃ¡lis Ã¼zenetben, vagy egy spam-ban. EzutÃ¡n meg kell hatÃ¡rozni, hogy milyen valÃ³szÃnÅ±sÃ©ggel kapunk normÃ¡lis Ã¼zenetet, Ã©s milyennel spam-et.

EzutÃ¡n, ha pl kÃvÃ¡ncsiak vagyunk, hogy egy szÃ³kombinÃ¡ciÃ³t tartalmazÃ³ email spam vagy nem spam, a szÃ³kombinÃ¡ciÃ³ban elÅfordulÃ³ szavak valÃ³szÃnÅ±sÃ©gÃ©t Ã¶ssze kell szorozni, majd megszorozni azzal, hogy milyen valÃ³szÃnÅ±sÃ©ggel kaptunk normÃ¡lis emailt, Ã©s milyennel spam-et. Amelyik valÃ³szÃnÅ±sÃ©gre nagyobb Ã©rtÃ©ket kapunk, abba az osztÃ¡lyba soroljuk a szÃ³kombinÃ¡ciÃ³t tartalmazÃ³ Ã¼zenetet.

ModellillesztÃ©s

LineÃ¡ris regressziÃ³?

MestersÃ©ges neuronhÃ¡lÃ³k

A mestersÃ©ges neuron a kÃ¶vetkezÅkÃ©ppen Ã©pÃ¼l fel

- bemeneti Ã©rtÃ©kek, valamilyen sÃºlyokkal megszorozva
- w0 bias weight, eltolÃ¡ssÃºly
- elÅszÃ¶r minden bemeneti Ã©rtÃ©ket megszorozza a hozzÃ¡ tartozÃ³ sÃºllyal, ezeket Ã¶sszeadja, majd kivonja belÅle az eltolÃ¡ssÃºlyt
- majd a kapott Ã©rtÃ©ken alkalmazzuk az aktivÃ¡ciÃ³s fÃ¼ggvÃ©nyt

Az aktivÃ¡ciÃ³s fÃ¼ggvÃ©ny cÃ©lja, hogy 1-hez kÃ¶zeli Ã©rtÃ©ket adjon, ha jÃ³ input Ã©rkezik, Ã©s 0-hoz kÃ¶zelit, ha rossz.

PÃ©lda aktivÃ¡ciÃ³s fÃ¼ggvÃ©nyek:

- kÃ¼szÃ¶bfÃ¼ggvÃ©ny: 0, ha az input <= mint 0, 1, ha nagyobb (perceptron)
- szigmoid fÃ¼ggvÃ©ny: g(x) = 1/(1 +e^âx)
- Rectifier aktivÃ¡ciÃ³: g(x) = max(0, x) (ReLU)

NeuronokbÃ³l hÃ¡lÃ³zatokat szokÃ¡s Ã©pÃteni. Egy hÃ¡lÃ³zatnak lehet tÃ¶bb rÃ©tege is. Van egy input, egy output Ã©s lehet tÃ¶bb rejtett rÃ©tege is. Egy rÃ©tegen belÃ¼l a neuronok kÃ¶zÃ¶tt nincs kapcsolat, csak a rÃ©tegek kÃ¶zÃ¶tt (elÅrecsatolt hÃ¡lÃ³zatok).

k-legkÃ¶zelebbi szomszÃ©d mÃ³dszere

13. LP alapfeladat, pÃ©lda, szimplex algoritmus, az LP geometriÃ¡ja, generÃ¡lÃ³elem vÃ¡lasztÃ¡si szabÃ¡lyok, kÃ©tfÃ¡zisÃº szimplex mÃ³dszer, speciÃ¡lis esetek (ciklizÃ¡ciÃ³-degenerÃ¡ciÃ³, nem korlÃ¡tos feladat, nincs lehetsÃ©ges megoldÃ¡s)

LP alapfeladat

@kÃ©p

LP alapfeladat: KeressÃ¼k adott lineÃ¡ris cÃ©lfÃ¼ggvÃ©ny szÃ©lsÅÃ©rtÃ©kÃ©t, Ã©rtelmezÃ©si tartomÃ¡nyÃ¡nak adott lineÃ¡ris korlÃ¡tokkal meghatÃ¡rozott rÃ©szÃ©ben. LehetsÃ©ges megoldÃ¡s: olyan p vektor, hogy p-t behelyettesÃtve x-be kielÃ©gÃti a feladat feltÃ©telrendszerÃ©t. LehetsÃ©ges megoldÃ¡si tartomÃ¡ny: az Ã¶sszes lehetsÃ©ges megoldÃ¡s halmaza. OptimÃ¡lis megoldÃ¡s: egy olyan lehetsÃ©ges megoldÃ¡s, ahol a cÃ©lfÃ¼ggvÃ©ny felveszi a maximumÃ¡t/minimumÃ¡t.

PÃ©lda: @kÃ©p

Szimplex algoritmus

Ahhoz, hogy lecserÃ©ljÃ¼k az egyenlÅtlensÃ©geket egyenlÅsÃ©gekre az LP alapfeladatban, adjunk hozzÃ¡ minden egyenlÅtlensÃ©g bal oldalÃ¡hoz egy mestersÃ©ges vÃ¡ltozÃ³t.

EzutÃ¡n fejezzÃ¼k ki a mestersÃ©ges vÃ¡ltozÃ³kat az egyenlet Ã¡trendezÃ©sÃ©vel.

A kapott egyenletrendszert hÃvjuk *szÃ³tÃ¡r alak*nak.

TermÃ©szetes vÃ¡ltozÃ³k: az eredeti vÃ¡ltozÃ³k MestersÃ©ges vÃ¡ltozÃ³k: az Ãºjonan felvett nemnegatÃv vÃ¡ltozÃ³k BÃ¡zisvÃ¡ltozÃ³k: a szÃ³tÃ¡r alakban bal oldalt Ã¡llÃ³ vÃ¡ltozÃ³k NembÃ¡zis vÃ¡ltozÃ³k: a szÃ³tÃ¡r alakban jobb oldalt Ã¡llÃ³ vÃ¡ltozÃ³k SzÃ³tÃ¡r bÃ¡zismegoldÃ¡sa: olyan x vektor, amelyben a szÃ³tÃ¡r nembÃ¡zis vÃ¡ltozÃ³inak Ã©rtÃ©ke nulla, Ãgy a bÃ¡zisvÃ¡ltozÃ³k Ã©rtÃ©ke a jobboldali konstans lesz LehetsÃ©ges (fizibilis) bÃ¡zismegoldÃ¡s: olyan bÃ¡zismegoldÃ¡s, ami egyben lehetsÃ©ges megoldÃ¡s is

A szimplex algoritmus:

- iteratÃv optimum keresÃ©s
- ismÃ©telt Ã¡ttÃ©rÃ©s mÃ¡s szÃ³tÃ¡rakra, a kÃ¶vetkezÅ feltÃ©telek betartÃ¡sa mellett:
    - minden iterÃ¡ciÃ³ szÃ³tÃ¡ra ekvivalens az Åt megelÅzÅÃ©vel
    - minden iterÃ¡ciÃ³ bÃ¡zismegoldÃ¡sÃ¡n a cÃ©lfÃ¼ggvÃ©ny Ã©rtÃ©ke nagyobb vagy egyenlÅ, mint az elÅzÅ iterÃ¡ciÃ³ban
    - minden iterÃ¡ciÃ³ bÃ¡zismegoldÃ¡sa lehetsÃ©ges megoldÃ¡s

Mi alapjÃ¡n tÃ©rjÃ¼nk Ã¡t mÃ¡sik szÃ³tÃ¡rra? Hogyan tÃ©rjÃ¼nk Ã¡t, hogy a feltÃ©telek teljesÃ¼ljenek? Honnan tudjuk, ha az aktuÃ¡lis bÃ¡zismegoldÃ¡s optimÃ¡lis? LÃ©tezik-e minden LP feladatnak optimuma?

Pivot lÃ©pÃ©s: Ãºj szÃ³tÃ¡r megadÃ¡sa egy bÃ¡zis Ã©s nembÃ¡zis vÃ¡ltozÃ³ szerepÃ©nek felcserÃ©lÃ©sÃ©vel BelÃ©pÅvÃ¡ltozÃ³: az a nembÃ¡zis vÃ¡ltozÃ³, ami a kÃ¶vetkezÅ szÃ³tÃ¡rra Ã¡ttÃ©rÃ©skor bÃ¡zisvÃ¡ltozÃ³vÃ¡ vÃ¡lik KilÃ©pÅ vÃ¡ltozÃ³: az a bÃ¡zisvÃ¡ltozÃ³, ami a kÃ¶v. szÃ³tÃ¡rra Ã¡ttÃ©rÃ©skor nembÃ¡zissÃ¡ vÃ¡lik SzÃ³tÃ¡rak ekvivalenciÃ¡ja: kÃ©t szÃ³tÃ¡r ekvivalens, ha az Ã¡ltaluk leÃrt egyenletrendszer Ã¶sszes lehetsÃ©ges megoldÃ¡sa Ã©s a hozzÃ¡juk tartozÃ³ cÃ©lfÃ¼ggvÃ©nyÃ©rtekek rendre megegyeznek

Pivot lÃ©pÃ©s elÅtti Ã©s utÃ¡ni szÃ³tÃ¡rak ekvivalensek.

SZIMPLEX:

- ha adott szÃ³tÃ¡rban minden cÃ©lfÃ¼ggvÃ©ny egyÃ¼tthatÃ³ negatÃv, akkor az aktuÃ¡lis bÃ¡zismegoldÃ¡s optimÃ¡lis
- ha nem, vÃ¡lasszuk a nembÃ¡zis vÃ¡ltozÃ³k kÃ¶zÃ¼l belÃ©pÅvÃ¡ltozÃ³nak valamely k.-at, amelyre a k. cÃ©lfÃ¼ggvÃ©ny egyÃ¼tthatÃ³ pozitÃv

- ha ennek a vÃ¡ltozÃ³nak minden egyenletben az egyÃ¼tthatÃ³ja negatÃv, a feladat nem korlÃ¡tos, megÃ¡llunk
- ha nem, akkor vÃ¡lasszuk az l. pozitÃv egyÃ¼tthatÃ³t, amelyre a konstans/egyÃ¼tthatÃ³ abszolÃºt Ã©rtÃ©ke minimÃ¡lis
- hajtsunk vÃ©gre egy pivot lÃ©pÃ©st Ãºgy, hogy xk legyen a belÃ©pÅvÃ¡ltozÃ³, Ã©s az l. feltÃ©tel bÃ¡zisvÃ¡ltozÃ³ja legyen a kilÃ©pÅ vÃ¡ltozÃ³

GenerÃ¡lÃ³elem vÃ¡lasztÃ¡si szabÃ¡lyok

Klasszikus szimplex pivot szabÃ¡ly:

- a lehetsÃ©ges belÃ©pÅvÃ¡ltozÃ³k kÃ¶zÃ¼l vÃ¡lasszuk a legnagyobb c_k Ã©rtÃ©kÅ±t, tÃ¶bb ilyen esetÃ©n a legkisebb indexÅ±t
- a lehetsÃ©ges kilÃ©pÅvÃ¡ltozÃ³k kÃ¶zÃ¼l vÃ¡lasszuk a legkisebb l indexÅ± egyenlet vÃ¡ltozÃ³jÃ¡t

Bland szabÃ¡ly

- a lehetsÃ©ges belÃ©pÅvÃ¡ltozÃ³k kÃ¶zÃ¼l vÃ¡lasszuk a legkisebb indexÅ±t
- a lehetsÃ©ges vÃ¡ltozÃ³k kÃ¶zÃ¼l vÃ¡lasszuk a legkisebb indexÅ±t

Legnagyobb nÃ¶vekmÃ©ny

Lexikografikus szabÃ¡ly

- kiegÃ©szÃtjÃ¼k epszilonokkal mestersÃ©gesen a szÃ³tÃ¡rat
- a lehetsÃ©ges belÃ©pÅvÃ¡ltozÃ³k kÃ¶zÃ¼l a legnagyobb c_k Ã©rtÃ©kÅ±t vÃ¡lasszuk, tÃ¶bb ilyennÃ©l a legkisebb indexÅ±t
- a lehetsÃ©ges kilÃ©pÅvÃ¡ltozÃ³k kÃ¶zÃ¼l azt, amelynek l indexÅ± egyenletÃ©re az egyÃ¼tthatÃ³kbÃ³l Ã¡llÃ³ vektor lexikografikusan a legkisebb

VÃ©letlen pivot

- 1 valÃ³szÃnÅ±sÃ©ggel terminÃ¡l

Az lp geometriÃ¡ja

ÃbrÃ¡zolhatjuk pl a lehetsÃ©ges megoldÃ¡sok halmazÃ¡t koordinÃ¡ta rendszerben, kÃ©t vÃ¡ltozÃ³ esetÃ©n.

Minden feltÃ©tel egy egyenest hatÃ¡roz meg, ezeket berajzoljuk. Ezzel valamilyen sokszÃ¶get kapunk meg, ennek a sokszÃ¶gnek a csÃºcsainak a koordinÃ¡tÃ¡i lesznek a lehetsÃ©ges megoldÃ¡sok.

KÃ©tfÃ¡zisÃº szimplex mÃ³dszer

Ha minden konstans nemnegatÃv az LP feladatban, akkor mehet a szimplex

De mi van, ha vannak negatÃv konstansok is?

VegyÃ¼nk egy segÃ©dfeladatot

- bevezetÃ¼nk egy Ãºj, x0 segÃ©dvÃ¡ltozÃ³t
- legyen w az Ãºj cÃ©lfÃ¼ggvÃ©ny, w=-x0
- tÃ©rjÃ¼nk Ã¡t szÃ³tÃ¡r alakra
- vegyÃ¼k a legnegatÃvabb jobboldalÃº egyenletet, Ã©s ebbÅl fejezzÃ¼k ki x0-t
- a tÃ¶bbibÅl a mestersÃ©ges vÃ¡ltozÃ³kat
- ezutÃ¡n mÃ¡r egy lehetsÃ©ges indulÃ³szÃ³tÃ¡rat kapunk

A standard feladatnak csak akkor lÃ©tezik lehetsÃ©ges megoldÃ¡sa, ha w=0 a hozzÃ¡ felÃrt segÃ©dfeladat optimuma.

Ha a segÃ©dfeladatot megoldjuk a szimplexszel, Ã©s annak optimuma 0, akkor a megoldÃ¡s utolsÃ³ szÃ³tÃ¡rÃ¡bÃ³l kÃ¶nynen felÃrhatunk egy olyan szÃ³tÃ¡rat, amely az eredeti feladat szÃ³tÃ¡ra, Ã©s bÃ¡zismegoldÃ¡sa lehetsÃ©ges megoldÃ¡s is egyben.

A szÃ³tÃ¡r felÃrÃ¡sÃ¡nak lÃ©pÃ©sei:

- az x0 = 0 feltÃ©telt elhagyjuk
- ha x0 bÃ¡zisvÃ¡ltozÃ³, akkor az egyenletÃ©nek jobb oldalÃ¡n lÃ©vÅ nem 0 egyÃ¼tthatÃ³jÃº vÃ¡ltozÃ³k egyikÃ©vel vÃ©grehajtunk egy pivot lÃ©pÃ©st
- elhagyjuk x0 megmaradt elÅforrÃ¡sait
- a cÃ©lfÃ¼ggvÃ©ny egyenletÃ©t lecserÃ©ljÃ¼k az eredeti cÃ©lfÃ¼ggvÃ©nyre, amit Ã¡tÃrunk az aktuÃ¡lis bÃ¡zisvÃ¡ltozÃ³knak megfelelÅen

A kÃ¶vetkezÅ fÃ¡zisban pedig az Ã¡tÃrt szÃ³tÃ¡ron futtatjuk a szimplex algoritmust

SpeciÃ¡lis esetek

CiklizÃ¡ciÃ³

DegenerÃ¡lt iterÃ¡ciÃ³s lÃ©pÃ©s: olyan szimplex iterÃ¡ciÃ³, amelyben nem vÃ¡ltozik a bÃ¡zismegoldÃ¡s DegenerÃ¡lt bÃ¡zismegoldÃ¡s: olyan bÃ¡zismegoldÃ¡s, amelyben egy vagy tÃ¶bb bÃ¡zisvÃ¡ltozÃ³ Ã©rtÃ©ke is 0

CiklizÃ¡ciÃ³: ha a szimplex algoritmus valamely iterÃ¡ciÃ³ja utÃ¡n egy korÃ¡bbi szÃ³tÃ¡rat visszakapunk, akkor az a ciklizÃ¡ciÃ³

Ha a szimplex algoritmus nem Ã¡ll meg, akkor ciklizÃ¡l! A ciklizÃ¡ciÃ³ elkerÃ¼lhetÅ megfelelÅ pivot szabÃ¡ly alkalmazÃ¡sÃ¡val (lexikografikus, Bland szabÃ¡ly) A ciklizÃ¡ciÃ³ oka a degenerÃ¡ciÃ³, azaz a bÃ¡zisvÃ¡ltozÃ³k 0-vÃ¡ vÃ¡lÃ¡sa a bÃ¡zismegoldÃ¡sban

Nem korlÃ¡tos

Ha az LP feladat maximalizÃ¡landÃ³/minimalizÃ¡landÃ³, Ã©s a cÃ©lfÃ¼ggvÃ©nye tetszÅlegesen nagy/kicsi Ã©rtÃ©ket felvehet, akkor nem korlÃ¡tos a feladat.

Nincs lehetsÃ©ges megoldÃ¡s

Ha a standard alakÃº LP feladatot kÃ©tfÃ¡zisÃº szimplex mÃ³dszerrel oldjuk meg, az elsÅ fÃ¡zis eldÃ¶nti, hogy van-e lehetsÃ©ges megoldÃ¡s.

Ha a felÃrt segÃ©dfeladatban az optimum Ã©rtÃ©ke kisebb, mint nulla, akkor nincs lehetsÃ©ges megoldÃ¡s, ha 0, akkor van.# 14. PrimÃ¡l-duÃ¡l feladatpÃ¡r, dualitÃ¡si komplementaritÃ¡si tÃ©telek, egÃ©sz Ã©rtÃ©kÅ± feladatok Ã©s jellemzÅik, a branch and bound mÃ³dszer, a hÃ¡tizsÃ¡k feladat

PrimÃ¡l-duÃ¡l feladatpÃ¡r

A primÃ¡l feladat

- maximalizÃ¡lunk
- c^T a cÃ©lfÃ¼ggvÃ©ny egyÃ¼tthatÃ³inak a vektora
- A az egyÃ¼tthatÃ³k mÃ¡trixa
- b a konstansok vektora

A duÃ¡l feladat

- minimalizÃ¡lunk
- b^T a cÃ©lfÃ¼ggvÃ©ny egyÃ¼tthatÃ³inak a vektora
- A^T az egyÃ¼tthatÃ³k mÃ¡trixa
- c a konstansok vektora
- <=-ket >=-re cserÃ©ljÃ¼k

A duÃ¡l feladat duÃ¡lisa az eredeti primÃ¡l feladat

TFH az LP feladatunk egy korlÃ¡tozott erÅforrÃ¡sok mellett maximÃ¡lis nyeresÃ©get cÃ©lzÃ³ gyÃ¡rtÃ¡si folyamat modellje. A duÃ¡l feladat megoldÃ¡sÃ¡ban az optimÃ¡lis megoldÃ¡s a primÃ¡l feladat i. erÅforrÃ¡sÃ¡hoz tartozÃ³ marginÃ¡lis Ã¡r/Ã¡rnyÃ©kÃ¡r, azaz az erÅforrÃ¡s Ã©rtÃ©ke az LP megoldÃ³jÃ¡nak szemszÃ¶gÃ©bÅl Ha tÃºl sok van egy erÅforrÃ¡sbÃ³l, az nem Ã©rhet tÃºl sokat TovÃ¡bbÃ¡ y_i*-nÃ¡l tÃ¶bbet nem Ã©rdemes fizetni az i. erÅforrÃ¡sÃ©rt, kevesebbet igen

DualitÃ¡si komplementaritÃ¡si tÃ©telek

Gyenge dualitÃ¡s

Ha x egy lehetsÃ©ges megoldÃ¡sa a primÃ¡l feladatnak, Ã©s y egy lehetsÃ©ges megoldÃ¡sa a duÃ¡l feladatnak, akkor a duÃ¡lis feladat bÃ¡rmely lehetsÃ©ges megoldÃ¡sa felsÅ korlÃ¡tja a primÃ¡l bÃ¡rmely lehetsÃ©ges megoldÃ¡sÃ¡nak (azaz az optimÃ¡lis megoldÃ¡snak is) (azaz a duÃ¡lis feladat bÃ¡rmely lehetsÃ©ges megoldÃ¡sa nagyobb vagy egyenlÅ a primÃ¡l bÃ¡rmely lehetsÃ©ges megoldÃ¡sÃ¡nÃ¡l)

A korlÃ¡tossÃ¡g Ã©s a megoldhatÃ³sÃ¡g nem fÃ¼ggetlenek egymÃ¡stÃ³l

Ha a primÃ¡l nem korlÃ¡tos, akkor a duÃ¡lnak nincs lehetsÃ©ges megoldÃ¡sa Ã©s fordÃtva. Lehet, hogy egyiknek sincs lehetsÃ©ges megoldÃ¡sa. Ha mindkettÅnek van, akkor mindkettÅ korlÃ¡tos A primÃ¡l Ã©s a duÃ¡l feladat egyidejÅ± optimalitÃ¡sa ellenÅrizhetÅ.

ErÅs dualitÃ¡s

Ha x *egy optimÃ¡lis megoldÃ¡sa a primÃ¡lnak, Ã©s y* egy optimÃ¡lis megoldÃ¡sa a duÃ¡l feladatnak, akkor c^Tx = *b^Ty*.

Ha valamelyik i. feltÃ©tel egyenlet nem Ã©les, azaz nem pontosan egyenlÅ a kÃ©t oldal, akkor a kapcsolÃ³dÃ³ duÃ¡l vÃ¡ltozÃ³ biztosan 0. Ha egy primÃ¡l vÃ¡ltozÃ³ pozitÃv, akkor a kapcsolÃ³dÃ³ duÃ¡lis feltÃ©tel biztosan Ã©les.

EgÃ©sz Ã©rtÃ©kÅ± feladatok Ã©s jellemzÅik

Tiszta egÃ©szÃ©rtÃ©kÅ± feladat (Integer Programming)

- Minden vÃ¡ltozÃ³nak egÃ©sznek kell lennie a megoldÃ¡sban.

Vegyes egÃ©sztÃ©rtÃ©kÅ± programozÃ¡si feladat (Mixed Integer Programming)

- Csak nÃ©hÃ¡ny vÃ¡ltozÃ³ra kÃ¶veteljÃ¼k meg, hogy egÃ©sz legyen

0-1 IP

- minden vÃ¡ltozÃ³ Ã©rtÃ©ke csak vagy 0 vagy 1 lehet

LP lazÃtÃ¡s

EgÃ©szÃ©rtÃ©kÅ± programozÃ¡si feladat LP lazÃtÃ¡sa az az LP, amelyet Ãºgy kapunk, hogy a vÃ¡ltozÃ³kra tett minden egÃ©szÃ©rtÃ©kÅ±sÃ©gi vagy 0-1 megkÃ¶tÃ©st eltÃ¶rlÃ¼nk.

- BÃ¡rmelyik IP lehetsÃ©ges megoldÃ¡shalmaza rÃ©sze az LP lazÃtÃ¡s lehetsÃ©ges megoldÃ¡startomÃ¡nyÃ¡nak
- MaximalizÃ¡lÃ¡snÃ¡l az LP lazÃtÃ¡s optimum Ã©rtÃ©ke nagyobbegyenlÅ, mint az IP optimumÃ©rtÃ©ke
- Ha az LP lazÃtÃ¡s lehetÃ©sges megoldÃ¡shalmazÃ¡nak minden csÃºcspontja egÃ©sz, akkor van egÃ©sz optimÃ¡lis megoldÃ¡sa, ami az IP megoldÃ¡sa is egyben
- Az LP lazÃtÃ¡s optimÃ¡lis megoldÃ¡sa bÃ¡rmilyen messze lehet az IP megoldÃ¡sÃ¡tÃ³l.

Branch and bound mÃ³dszer

1. lÃ©pÃ©s

Megoldjuk az LP lazÃtÃ¡st, ha a megoldÃ¡s egÃ©szÃ©rtÃ©kÅ±, akkor done

1. lÃ©pÃ©s

Ha van lezÃ¡ratlan rÃ©szfeladatunk, akkor azt egy xi nem egÃ©sz vÃ¡ltozÃ³ szerint kÃ©t rÃ©szfeladatra bontjuk. Ha xi Ã©rtÃ©ke xi, *akkor xi <= floor(xi*) Ã©s xi>= ceil(xi*) feltÃ©teleket vesszÃ¼k hozzÃ¡ egy egy rÃ©szfeladatunkhoz

- a rÃ©szproblÃ©mÃ¡kat egy fÃ¡ba rendezzÃ¼k
- a gyÃ¶kÃ©r az elsÅ rÃ©szfeladat, az LP lazÃtÃ¡s
- a leszÃ¡rmazottai az Ã¡gaztatott rÃ©szproblÃ©mÃ¡k
- a hozzÃ¡vett feltÃ©teleket az Ã©leken adjuk meg
- a csÃºcsokban jegyezzÃ¼k az LP-k optimÃ¡lis megoldÃ¡sait

Lehet, hogy olyan rÃ©szproblÃ©mÃ¡t kapunk, aminek nincs lehetsÃ©ges megoldÃ¡sa, ekkor ezt a levelet elhagyjuk TalÃ¡lhatunk megoldÃ¡sjelÃ¶lteket is, ezek alsÃ³ korlÃ¡tok az eredeti IP optimÃ¡lis Ã©rtÃ©kÃ©re. Ha talÃ¡lunk korÃ¡bbi megoldÃ¡sjelÃ¶ltnÃ©l jobb megoldÃ¡st, akkor a rosszabbat elvetjÃ¼k.

Egy csÃºcs felderÃtett/lezÃ¡rt, ha

- nincs lehetsÃ©ges megoldÃ¡sa
- megoldÃ¡sa egÃ©szÃ©rtÃ©kÅ±
- felderÃtettÃ¼nk mÃ¡r olyan egÃ©sz megoldÃ¡st, ami jobb a rÃ©szfeladat megoldÃ¡sÃ¡nÃ¡l

Egy rÃ©szfeladatot kizÃ¡runk, ha

- nincs lehetsÃ©ges megoldÃ¡sa
- felderÃtettunk mÃ¡r olyan egÃ©sz megoldÃ¡st, ami jobb a rÃ©szfeladat megoldÃ¡sÃ¡nÃ¡l

A hÃ¡tizsÃ¡k feladat

Egy olyan IP-t, amiben csak egy feltÃ©tel van, hÃ¡tizsÃ¡k feladatnak nevezÃ¼nk. Van egy hÃ¡tizsÃ¡kunk egy fix kapacitÃ¡ssal, Ã©s tÃ¡rgyaink, Ã©rtÃ©kekkel Ã©s sÃºlyokkal megadva.

MaximalizÃ¡lni akarjuk a tÃ¡skÃ¡ba rakott tÃ¡rgyak Ã©rtÃ©kÃ©t, Ãºgy hogy a benne lÃ©vÅ tÃ¡rgyak nem haladhatjÃ¡k meg a hÃ¡tizsÃ¡k kapacitÃ¡sÃ¡t persze. 0-1 IP feladat, egy tÃ¡rgyat viszÃ¼nk vagy nem

Az LP lazÃtÃ¡s kÃ¶nnyen szÃ¡mÃthatÃ³, a relatÃv hasznossÃ¡g szerint tesszÃ¼k a tÃ¡rgyakat a tÃ¡skÃ¡ba, vagyis az Ã©rtÃ©k/sÃºly hÃ¡nyadosuk szerint. Branch and bound mÃ³dszerrel ez is megoldhatÃ³

Legrosszabb esetben 2^n rÃ©szfeladatot kell megoldani, NP nehÃ©z a feladat. EgÃ©szÃ©rtÃ©kÅ±nÃ©l mÃ©g rosszabb, 2^Mn, ahol M a lehetsÃ©ges egÃ©szek szÃ¡ma egy vÃ¡ltozÃ³ra

15. Processzusok, szÃ¡lak/fonalak, processzus lÃ©trehozÃ¡sa/befejezÃ©se, processzusok Ã¡llapotai, processzus leÃrÃ¡sa. Ãtemezési stratÃ©giÃ¡k Ã©s algoritmusok kÃ¶tegelt, interaktÃv Ã©s valÃ³s idejÅ± rendszereknÃ©l, Ã¼temezÃ©si algoritmusok cÃ©ljai. Kontextus-csere

OperÃ¡ciÃ³s rendszer

A szÃ¡mÃtÃ³gÃ©peknek azt az alapprogramjÃ¡t, mely kÃ¶zvetlenÃ¼l kezeli a hardvert, Ã©s egy egysÃ©ges kÃ¶rnyezetet biztosÃt a szÃ¡mÃtÃ³gÃ©pen futtatandÃ³ alkalmazÃ¡soknak, op.rendszernek nevezzÃ¼k. Egy moder szÃ¡mÃtÃ³gÃ©p a kÃ¶vetkezÅkbÅl Ã¡ll:

- egy vagy tÃ¶bb processzor
- memÃ³ria
- lemezek
- I/O eszkÃ¶zÃ¶k
- â¦

Ezen komponensek kezelÃ©se egy szoftver rÃ©teget igÃ©nyel â Ez az op. rendszer Feladatai:

- felhasznÃ¡lÃ³ kÃ©nyelmÃ©nek, vÃ©delmÃ©nek biztosÃtÃ¡sa
- egy egysÃ©ges kÃ¶rnyezetet biztosÃt a szÃ¡mÃtÃ³gÃ©pen futtatandÃ³ alkalmazÃ¡soknak
- a rendszer hatÃ©konysÃ¡gÃ¡nak, teljesÃtmÃ©nyÃ©nek maximalizÃ¡lÃ¡sa = erÅforrÃ¡sok kezelÃ©se
- a programok vÃ©grehajtÃ¡sÃ¡t vezÃ©rli
- biztosÃtja a felhasznÃ¡lÃ³ Ã©s a szÃ¡mÃtÃ³gÃ©pes rendszer kÃ¶zÃ¶tti kommunikÃ¡ciÃ³t

FelÃ©pÃtÃ©se:

- RendszerhÃ©j (shell)
  - Feladata a parancsÃ©rtelmezÃ©s.
  - Lehet a shell parancssoros (CLI - Command Line Interface - mint, pl. DOS), vagy grafikus felÃ¼letÅ±
  - KapcsolattartÃ¡s a felhasznÃ¡lÃ³val
- Alacsony szintÅ± segÃ©dprogramok
  - felhasznÃ¡lÃ³i "Ã©lmÃ©nyt" fokozÃ³ kiegÃ©szÃtÅ programok (pl szÃ¶vegszerkesztÅk, fordÃtÃ³programok), amelyek nem kÃ©pzik a rendszer elvÃ¡laszthatatlan rÃ©szÃ©t
- Kernel
  - az operÃ¡ciÃ³s rendszer alapja (magja), amely felelÅs a hardver erÅforrÃ¡sainak kezelÃ©sÃ©Ã©rt)
  - kÃ¶zvetlenÃ¼l a hardverrel Ã¡ll kapcsolatban.
  - Ki- Ã©s bemeneti eszkÃ¶zÃ¶k kezelÃ©se (billentyÅ±zet, monitor stb.)
  - Programok, folyamatok futÃ¡sÃ¡nak kezelÃ©se
    - IndÃtÃ¡s, futÃ¡si feltÃ©telek biztosÃtÃ¡sa, leÃ¡llÃtÃ¡s
    - MemÃ³ria-hozzÃ¡fÃ©rÃ©s biztosÃtÃ¡sa
    - Processzor idejÃ©nek elosztÃ¡sa

Az operÃ¡ciÃ³s rendszerek csoportosÃtÃ¡sa

- FelhasznÃ¡lÃ³k szÃ¡ma szerint:
  - egy felhasznÃ¡lÃ³s pl.: DOS, Win 9x
  - tÃ¶bb felhasznÃ¡lÃ³s pl. Linux, Win NT
- Hardver mÃ©rete szerint:
  - kisgÃ©pes (UNIX)
  - nagygÃ©pes (Main Frame, Cray - szuper szÃ¡mÃtÃ³gÃ©p)
  - mikrogÃ©pes (DOS, WIN 9X, UNIX)
- ProcesszorkezelÃ©s szerint:
  - egy feladatos (DOS)
  - tÃ¶bb feladatos (WIN 9X, WIN NT, UNIX)
- CÃ©l szerint:

- Ã¡ltalÃ¡nos (DOS, WIN 9X, WIN NT, UNIX)
- speciÃ¡lis (folyamatvezÃ©rlÅ operÃ¡ciÃ³s rendszerek)

- OperÃ¡ciÃ³s rendszer felÃ©pÃtÃ©se szerint:
  - monolitikus

    ## - A monolitikus operÃ¡ciÃ³s rendszer (mint pÃ©ldÃ¡ul a UNIX) magja egyetlen programbÃ³l Ã¡ll. Ebben a programban az eljÃ¡rÃ¡sok szabadon hÃ-vhatjÃ¡k egymÃ¡st, a kÃ¶ztÃ¼k levo kommunikÃ¡ciÃ³ eljÃ¡rÃ¡sparamÃ©tereken Ã©s globÃ¡lis vÃ¡ltozÃ³kon keresztÃ¼l zajlik.

  - rÃ©teges szerkezetÅ± (WIN NT, UNIX)
    - A rÃ©tegzett szerkezetu operÃ¡ciÃ³s rendszer magja tÃ¶bb modulbÃ³l Ã¡ll, Ã©s a modulok kÃ¶zÃ¶tt egy export-import hierarchia figyelheto meg: minden modul kizÃ¡rÃ³lag a hierarchiÃ¡ban alatta levo modul interfÃ©szÃ©t hasznÃ¡lja.
  - kliens/szerver felÃ©pÃtÃ©sÅ± - HÃ¡lÃ³zati operÃ¡ciÃ³s rendszer
    - a szerveren fut, Ã©s lehetÅvÃ© teszi a szervernek az adatok, felhasznÃ¡lÃ³k, csoportok, alkalmazÃ¡sok, a hÃ¡lÃ³zati biztonsÃ¡g Ã©s egyÃ©b hÃ¡lÃ³zati funkciÃ³k kezelÃ©sÃ©t.
    - A kliens/szerver hÃ¡lÃ³zati operÃ¡ciÃ³s rendszerek lehetÅvÃ© teszik funkciÃ³k Ã©s alkalmazÃ¡sok kÃ¶zpontosÃtÃ¡sÃ¡t egy vagy tÃ¶bb dedikÃ¡lt szerveren. A szerver a rendszer kÃ¶zpontja, engedÃ©lyezi az erÅforrÃ¡sokhoz valÃ³ hozzÃ¡fÃ©rÃ©st Ã©s biztonsÃ¡gos kapcsolatot nyÃºjt
  - vegyes
  - virtuÃ¡lis gÃ©pek
    - A virtuÃ¡lis gÃ©peken alapulÃ³ operÃ¡ciÃ³s rendszerben kÃ¶zponti rÃ©szen helyezkednek el a virtuÃ¡lis gÃ©peket menedzselo (hypervisor) rendszerrutinok. Ez a program lehetovÃ© teszi a hardver eroforrÃ¡sainak (CPU, diszk, perifÃ©riÃ¡k, memÃ³ria, ...) tÃ¶bb operÃ¡ciÃ³s rendszer kÃ¶zÃ¶tti hatÃ©kony elosztÃ¡sÃ¡t. A hypervisor leggyakrabban a szÃ¡mÃtÃ³gÃ©p hardverÃ©t "tÃ¶bbszÃ¶rÃ¶zi meg" Ãºgy, hogy a rajta futÃ³ operÃ¡ciÃ³s rendszerek azt higgyÃ©k, hogy Ã¶vÃ©kÃ© az egÃ©sz gÃ©p (pedig "csak" egy virtuÃ¡lis gÃ©pen futnak)
- A felhasznÃ¡lÃ³i felÃ¼let szerint:
  - szÃ¶veges (DOS, UNIX)
  - grafikus (WINDOWS)

Op. rendszer generÃ¡ciÃ³k

- ElsÅ generÃ¡ciÃ³s (1945-1955)

  - nincs os. leginkÃ¡bb csak hardver elemekbÅl Ã¡llt (kÃ¼lÃ¶nfÃ©le kapcsolÃ³k, cÃmkivÃ¡lasztÃ³ kulcsok, indÃtÃ³-, megÃ¡llÃ-tÃ³-, lÃ©pÃ©senkÃ©nti vÃ©grehajtÃ¡st kivÃ¡ltÃ³ gombok stb.
  - programozÃ³=gÃ©pkezelÅ=programok vÃ©grehajtÃ¡sÃ¡nak vezÃ©rlÅje
  - binÃ¡ris kÃ³dolÃ¡s

- MÃ¡sodik generÃ¡ciÃ³ (1955-1965)

  - os van
  - egyidejÅ±leg 1 processzus
  - fortran programozÃ¡s
- Harmadik generÃ¡ciÃ³s (1965-1980)
  - os van, szoftverrel megvalÃ³sÃtott operÃ¡ciÃ³s rendszer
  - integrÃ¡lt Ã¡ramkÃ¶rÃ¶k, multiprogramozÃ¡s
  - egyidejÅ±leg tÃ¶bb proc
  - CPU idÅszeletelÃ©s (time slicing): egy processzus egy meghatÃ¡rozott max idÅintervallumon keresztÃ¼l hasznÃ¡lhatja a CPU-t folyamatosan (ez a tmax idÅ). Ha ez letelik az op.rendszer processzus Ã¼temezÅ Ã¡tadja a CPU-t egy mÃ¡sik processusnak
  - Ãtmeneti tÃ¡rolÃ¡s (spooling): az I/O adatok elÅszÃ¶r gyors hÃ¡ttÃ©rtÃ¡rolÃ³kra kerÃ¼lnek, majd a processzus innen kapja/ide Ãrja az adatait
- Negyedik generÃ¡ciÃ³s (1980-tÃ³l)
  - szemÃ©lyi szÃ¡mÃtÃ³gÃ©pek
  - parancssoros, grafikus felÃ¼let
  - pc, workstation: egyetlen felhasznÃ¡lÃ³, egy idÅben tÃ¶bb feladat (Windows, MacOS)
  - hÃ¡lÃ³zati os: hÃ¡lÃ³zaton keresztÃ¼l tÃ¶bb felhasznÃ¡lÃ³, kapcsolÃ³dik, minden felhasznÃ¡lÃ³ tÃ¶bb feladatot futtathat (Unix, Linux)
  - osztott os: egy feladatot egy idÅben tÃ¶bb szÃ¡mÃtÃ³gÃ©pes rendszer vÃ©gez

SZERINTEM INNENTÅL KELL

Processzusok, szÃ¡lak/fonalak, processzus lÃ©trehozÃ¡sa/befejezÃ©se, processzusok

Ã¡llapotai, processzus leÃrÃ¡sa

Processzus

- A vÃ©grehajtÃ¡s alatt lÃ©vÅ program.
- SzekvenciÃ¡lisan vÃ©grehajtÃ³dÃ³ program
- EgyidejÅ±leg tÃ¶bb processus lÃ©tezik: A processzor idejÃ©t meg kell osztani az egyidejÅ±leg lÃ©tezÅ processzusok kÃ¶zÃ¶tt: idÅosztÃ¡s (time sharing)
- FutÃ³ processzusok is lÃ©trehozhatnak processzusokat: KooperatÃv folyamatok, egymÃ¡ssal egyÃ¼ttmÅ±kÃ¶dÅ, de amÃºgy fÃ¼ggetlen processzusok
- Az erÅforrÃ¡sokat az OS-tÅl kapjÃ¡k (centralizÃ¡lt erÅforrÃ¡s kezelÃ©s)
- jogosultsÃ¡gokkal rendelkeznek
- ElÅtÃ©rben Ã©s Ã¡ttÃ©rben futÃ³ folyamatok Processus Ã¡llapotok:
- FutÃ¡skÃ©sz: kÃ©szen Ã¡ll a futÃ¡sra, csak ideiglenesen le lett Ã¡llÃtva, hogy egy mÃ¡sik processzus futhasson
- FutÃ³: a proc bitrokolja a CPU-t
- Blokkolt: bizonyos kÃ¼lsÅ esemÃ©ny bekÃ¶vetkezÃ©sÃ©ig nem kÃ©pes futni
- IniciÃ¡lis
- TerminÃ¡lis
- FelfÃ¼ggesztett

ProcesszustÃ¡blÃ¡zat Ã©s PCB

A proc nyilvÃ¡ntartÃ¡sÃ¡ra, tulajdonsÃ¡gainak leÃrÃ¡sÃ¡ra szolgÃ¡lÃ³ memÃ³riaterÃ¼let. ProcessusonkÃ©nt egy egy bejegyzÃ©s - Processzus vezÃ©rlÅ blokk (PCB) PCB tartalma:

- azonosÃtÃ³: processzus id
- processzus Ã¡llapota
- CPU Ã¡llapota: a kontextus cserÃ©hez
- jogosultsÃ¡gok, prioritÃ¡s
- birtokolt erÅforrÃ¡sok

Processzus lÃ©trehozÃ¡sa

- FutÃ³ processzusok is lÃ©trehozhatnak processzusokat: KooperatÃv folyamatok, egymÃ¡ssal egyÃ¼ttmÅ±kÃ¶dÅ, de amÃºgy fÃ¼ggetlen processzusok
- EgyszerÅ± esetekben megoldhatÃ³, hogy minden processzus elÃ©rhetÅ az OS elindulÃ¡sa utÃ¡n
- ÃltalÃ¡nos cÃ©lÃº rendszerekben szÃ¼ksÃ©g van a processzusok lÃ©trehozÃ¡sÃ¡ra Ã©s megszÃ¼ntetÃ©sÃ©re
- Processzusokat lÃ©trehozÃ³ esemÃ©nyek:
  - Rendszer inicializÃ¡lÃ¡sa
  - FelhasznÃ¡lÃ³ Ã¡ltal kezdemÃ©nyezett
  - KÃ¶tegelt feladat kezdemÃ©nyezÃ©se
- Az OS indulÃ¡sakor sok processzus keletkezik:
  - FelhasznÃ¡lÃ³kkal tartjÃ¡k a kapcsolatot: ElÅtÃ©rben futnak
  - Nincsenek felhasznÃ¡lÃ³hoz rendelve:
    - SajÃ¡t feladatuk van
    - HÃ¡ttÃ©rben futnak
- LÃ©pÃ©sei:
  1. MemÃ³riaterÃ¼let foglalÃ¡sa a PCB szÃ¡mÃ¡ra
  2. PCB kitÃ¶ltÃ©se iniciÃ¡lis adatokkal
  3. ProgramszÃ¶veg, adatok, verem szÃ¡mÃ¡ra memÃ³riafoglalÃ¡s, betÃ¶ltÃ©s
  4. A PCB procok lÃ¡ncra fÅ±zÃ©se, futÃ¡skÃ©sz Ã¡llapot. EttÅl kezdve a proc osztozik a CPU-n.

Processzus befejezÃ©se

- SzabÃ¡lyos kilÃ©pÃ©s (exit(0)): Ã¶nkÃ©ntes, vÃ©gzett a feladatÃ¡val
- KilÃ©pÃ©s hiba miatt
- KilÃ©pÃ©s vÃ©gzetes hiba miatt: Ã¶nkÃ©ntelen, illegÃ¡lis utasÃtÃ¡s, nullÃ¡val osztÃ¡s
- Egy mÃ¡sik proc megsemmisÃti: Ã¶nkÃ©ntelen, mÃ©sik proc kill() utasÃtÃ¡sÃ¡ra
- LÃ©pÃ©sei: 1. Gyermek procok megszÃ¼ntetÃ©se (rekurzÃvan) 2. PCB procok lÃ¡ncrÃ³l valÃ³ levÃ©tele, terminÃ¡lis Ã¡llapot. EttÅl kezdve a proc nem osztocik a CPU-n 3. Proc bitrokÃ¡ban lÃ©vÅ erÅforrÃ¡sok felszabadÃtÃ¡sa (pl. fÃ¡jlok lezÃ¡rÃ¡sa) 4. A memÃ³riatÃ©rkÃ©pnek (konstansok, vÃ¡ltozÃ³k, dinamikus vÃ¡ltozÃ³k) megfelelÅ memÃ³riaterÃ¼let felszabadÃtÃ¡sa 5. PCB memÃ³riaterÃ¼letÃ©nek felszabadÃtÃ¡sa

SzÃ¡lak/fonalak (thread)

- ÃnÃ¡llÃ³ vÃ©grehajtÃ¡si egysÃ©gkÃ©nt mÅ±kÃ¶dÅ program, objektum, szekvenciÃ¡lisan vÃ©grehajthatÃ³ utasÃtÃ¡s-sorozat
- A proc hozza lÃ©tre (akÃ¡r tÃ¶bbet is egyszerre)
- Osztozik a lÃ©trehozÃ³ proc erÅforrÃ¡sain
- Egy folyamaton belÃ¼l tÃ¶bb tevÃ©kenysÃ©g vÃ©gezhetÅ pÃ¡rhuzamosan
- SzÃ¡lak megvalÃ³sÃtÃ¡sa:
    - A felhasznÃ¡lÃ³ kezeli a szÃ¡lakat egy fÃ¼ggvÃ©nykÃ¶nyvtÃ¡r segÃtsÃ©gÃ©vel. Ekkor a kernel (az operÃ¡ciÃ³s rendszer alapja (magja), amely felelÅs a hardver erÅforrÃ¡sainak kezelÃ©sÃ©Ã©rt) nem tud semmit a szÃ¡lakrÃ³l
    - A kernel kezeli a szÃ¡lakat. SzÃ¡lak lÃ©trehozÃ¡sa Ã©s megszÃ¼ntetÃ©se kernelhÃvÃ¡sokkal tÃ¶rtÃ©nik

ÃtemezÃ©si stratÃ©giÃ¡k Ã©s algoritmusok kÃ¶tegelt, interaktÃv Ã©s valÃ³s idejÅ± rendszereknÃ©l, Ã¼temezÃ©si algoritmusok cÃ©ljai

ÃtemezÅ

- Egy CPU Ã¡ll rendelkezÃ©sre. Processzusok versengenek a CPU-Ã©rt
- Az OS dÃ¶nti el, hogy melyik kapja meg a CPU-t
- Az Ã¼temezÅ (scheduler) hozza meg a dÃ¶ntÃ©st ï ÃtemezÃ©si algoritmus
- Feladata: Egy adott idÅpontban futÃ¡skÃ©sz procok kÃ¶zÃ¼l egy kivÃ¡lasztÃ¡sa, amely a kÃ¶vetkezÅkben a CPU-t bitrokolni fogja
- Mikor kell Ã¼temezni?: amikor egy processus befejezÅdik vagy blokkolÃ³dik
- CÃ©ljai:
    - a CPU legyen jÃ³l kihasznÃ¡lt
    - az Ã¡tfutÃ¡si idÅ (proc lÃ©trejÃ¶ttÃ©tÅl megszÅ±nÃ©sÃ©ig eltelt idÅ) legyen rÃ¶vid
    - egysÃ©gnyi idÅ alatt minÃ©l tÃ¶bb proc teljesÃ¼ljÃ¶n

ÃtemezÃ©s kÃ¶tegelt rendszerekben

A manapsÃ¡g hasznÃ¡latos op.rendszerek nem tartoznak a kÃ¶tegelt rendszerek (: ElÅre meghatÃ¡rozott sorrend szerint vÃ©grehajtandÃ³ feladatok egyÃ¼ttese.) vilÃ¡gÃ¡ba, mÃ©gis Ã©rdemes rÃ¶viden megemlÃteni ezek Ã¼temezÃ©si tÃpusait. - Sorrendi Ã¼temezÃ©s: - FutÃ¡sra kÃ©sz folyamatok egy vÃ¡rakozÃ³ sorban helyezkednek el. - A sorban levÅ elsÅ folyamatot hajtja vÃ©gre a kÃ¶zponti egysÃ©g. Ha befejezÅdik a folyamat vÃ©grehajtÃ¡sa, az Ã¼temezÅ a sorban kÃ¶vetkezÅ feladatot veszi elÅ. - Ãj feladatok a sor vÃ©gÃ©re kerÃ¼lnek - Ha az aktuÃ¡lisan futÃ³ folyamat blokkolÃ³dik, akkor a sorban kÃ¶vetkezÅ folyamat jÃ¶n, mÃg a blokkolt folyamat, ha Ãºjra futÃ¡sra kÃ©sz lesz, akkor a sor vÃ©gÃ©re kerÃ¼l, Ã©s majd idÅvel Ãºjra rÃ¡ kerÃ¼l a vezÃ©rlÃ©s. - LegrÃ¶videbb feladat elÅszÃ¶r: - az a folyamat kerÃ¼l elÅszÃ¶r Ã¼temezÃ©sre, melyiknek a legkisebb a futÃ¡si ideje. - az alkalmazhatÃ³sÃ¡g szempontjÃ¡bÃ³l nem ideÃ¡lis, ha nem tudjuk elÅre a folyamatok vÃ©grehajtÃ¡si idejÃ©t. - LegrÃ¶videbb maradÃ©k futÃ¡sidejÅ±: - Ismerni kell a folyamatok futÃ¡si idejÃ©t elÅre. - Amikor Ãºj folyamat Ã©rkezik, vagy a blokkolÃ¡s miatt egy kÃ¶vetkezÅ folyamathoz kerÃ¼l a vezÃ©rlÃ©s, akkor nem a teljes folyamat vÃ©grehajtÃ¡si idejÃ©t, hanem csak a hÃ¡tralÃ©vÅ idÅt vizsgÃ¡lja az Ã¼temezÅ, Ã©s amelyik folyamatnak legkisebb a maradÃ©k futÃ¡si ideje, az kerÃ¼l Ã¼temezÃ©sre

- HÃ¡romszintÅ± futÃ¡sidejÅ±:
    - A feladatok a kÃ¶zponti memÃ³riÃ¡ban vannak, kÃ¶zÃ¼lÃ¼k egyet hajt vÃ©gre a kÃ¶zponti egysÃ©g. ElÅfordulhat, hogy a tÃ¶bbi feladat kÃ¶zÃ¼l ki kell rakni egyet a hÃ¡ttÃ©rtÃ¡rba, mivel a mÅ±kÃ¶dÃ©s sorÃ¡n elfogyhat a memÃ³ria.
    - Az a dÃ¶ntÃ©st, hogy a futÃ¡sra jelentkezÅ folyamatok milyen sorrendben kerÃ¼ljenek be a memÃ³riÃ¡ba, a bebocsÃ¡tÃ³ Ã¼temezÅ hozza meg.

ÃtemezÃ©s interaktÃv rendszereknÃ©l

- Round Robin
    - Az Ã¼temezÅ beÃ¡llÃt egy idÅintervallumot egy idÅzÃtÅ segÃtsÃ©gÃ©vel Ã©s amikor az idÅzÃtÅ lejÃ¡r megszakÃtÃ¡st ad.
    - Megadott idÅkÃ¶zÃ¶nkÃ©nt Ã³ramegszakÃtÃ¡s kÃ¶vetkezik be Ã©s ekkor az Ã¼temezÅ a kÃ¶vetkezÅ folyamatnak adja a processzort.
    - A folyamatokat egy sorban tÃ¡rolja a rendszer, Ã©s amikor lejÃ¡rt az idÅszelet, akkor az a folyamat, amelyiktÅl az Ã¼temezÅ Ã©ppen elveszi a vezÃ©rlÃ©st, a sor vÃ©gÃ©re kerÃ¼l
- PrioritÃ¡sos Ã¼temezÃ©s
    - FelmerÃ¼l az igÃ©ny, hogy nem feltÃ©tlenÃ¼l egyformÃ¡n fontos minden egyes folyamat.
    - A folyamatokhoz egy fontossÃ¡gi mÃ©rÅszÃ¡mot, prioritÃ¡st (prioritÃ¡si osztÃ¡lyt) rendel hozzÃ¡
    - A legmagasabb prioritÃ¡sÃº futÃ¡skÃ©sz processzus kapja meg a CPU-t

ÃtemezÃ©s valÃ³s idejÅ± rendszereknÃ©l

AlapvetÅ szerepe van az idÅnek Ha a feladatainknak nemcsak azt szabjuk meg, hogy hajtÃ³djanak vÃ©gre valamilyen korrekt Ã¼temezÃ©s szerint, hanem az is egy kritÃ©rium, hogy egy adott kÃ©rÃ©st valamilyen idÅn belÃ¼l ki kell szolgÃ¡lni, akkor valÃ³s idejÅ± op.rendszerrÅl beszÃ©lÃ¼nk. A megfelelÅ hatÃ¡ridÅk betartÃ¡sa Ãºgy valÃ³sÃthatÃ³ meg, hogy egy programot tÃ¶bb folyamatra bontunk, Ã©s ezeknek a rÃ¶vid folyamatoknak az Ã¼temezÅ biztosÃtja a szÃ¡mukra elÅÃrt hatÃ¡ridÅ betartÃ¡sÃ¡t - SzigorÃº valÃ³s idejÅ± rendszer - a hatÃ¡ridÅ betartÃ¡sa kÃ¶telezÅ - TolerÃ¡ns valÃ³s idejÅ± (soft real-time) rendszer - a hatÃ¡ridÅk kis mulasztÃ¡sa mÃ©g elfogadhatÃ³, tolerÃ¡lhatÃ³.

Kontextus csere

Egy CPU van Ã©s tÃ¶bb egyidejÅ±leg lÃ©tezÅ processzus. A CPU vÃ¡ltakozva hajtja vÃ©gre a processzusokat. A kontextus csere, amikor a CPU Ã¡tvÃ¡lt P1 processzusrÃ³l a P2 processzusra. Ilyenkor P1 Ã¡llapotÃ¡t el kell menteni a CPU regisztereibÅl, az erre fenttartott memÃ³riaterÃ¼letre, majd P2 mentett Ã¡llapotÃ¡t vissza kell Ã¡llÃtani a CPU regisztereiben.

SZERINTEM INNENTÅL MÃR NEM KELL

OperÃ¡ciÃ³s rendszerek feladatai, fajtÃ¡i, felÃ©pÃtÃ©sei Ã©s felhasznÃ¡lÃ¡si terÃ¼letei. PÃ¡rhuzamossÃ¡ggal kapcsolatos fogalmak, problÃ©mÃ¡k Ã©s megoldÃ¡saik. Folyamatok, szÃ¡lak fogalma, megvalÃ³sÃtÃ¡saik Ã©s Ã¼temezÃ©si mÃ³dszereik. MemÃ³riakezelÃ©ssel, Ã¡llomÃ¡nyrendszerekkel Ã©s szolgÃ¡ltatÃ¡saikkal kapcsolatos fogalmak Ã©s megvalÃ³sÃtÃ¡si mÃ³dszereik

MemÃ³riakezelÃ©ssel, Ã¡llomÃ¡nyrendszerekkel Ã©s szolgÃ¡ltatÃ¡saikkal kapcsolatos fogalmak Ã©s megvalÃ³sÃtÃ¡si mÃ³dszereik MemÃ³riakezelÃ©s A memÃ³ria az egyik legfontosabb (Ã©s gyakran a legszÅ±kÃ¶sebb) erÅforrÃ¡s, amivel egy operÃ¡ciÃ³s rendszernek gazdÃ¡lkodnia kell; fÅleg a tÃ¶bbfelhasznÃ¡lÃ³s rendszerekben, ahol gyakran olyan sok Ã©s nagy folyamat fut, hogy egyÃ¼tt nem fÃ©rnek be egyszerre a memÃ³riÃ¡ba. A tÃ¶bbfeladatos feldolgozÃ¡s megjelenÃ©sÃ©vel azonban szÃ¼ksÃ©gessÃ© vÃ¡lt a memÃ³riÃ¡nak a futÃ³ folyamatok kÃ¶zÃ¶tti valamilyen âigazsÃ¡gosâ elosztÃ¡sÃ¡ra. - MultiprogramozÃ¡s megvalÃ³sÃtÃ¡sa rÃ¶gzÃtett memÃ³ria szeletekkel. - Osszuk fel a memÃ³riÃ¡t n szeletre. (Fix szeletek) pl. rendszerindÃtÃ¡snÃ¡l ez megtehetÅ - a fÅmemÃ³ria kihasznÃ¡lÃ¡sa nem jÃ³: minden program, mÃ©retÃ©tÅl fÃ¼ggetlenÃ¼l egy egÃ©sz partÃciÃ³t elfoglal. - MegoldÃ¡s: nem egyenlÅ mÃ©retÅ± partÃciÃ³k - MultiprogramozÃ¡s megvalÃ³sÃtÃ¡sa memÃ³ria csere hasznÃ¡lattal. - Teljes folyamat mozgatÃ¡sa memÃ³ria-lemez kÃ¶zÃ¶tt - Nincs rÃ¶gzÃtett memÃ³ria partÃciÃ³, mindegyik dinamikusan vÃ¡ltozik, ahogy az op.rendszer odavissza rakosgatja a folyamatokat. - Dinamikus, jobb memÃ³ria kihasznÃ¡ltsÃ¡gÃº lesz a rendszer, de a sok csere lyukakat hoz lÃ©tre! ï MemÃ³ria tÃ¶mÃ¶rÃtÃ©st kell vÃ©gezni - MultiprogramozÃ¡s megvalÃ³sÃtÃ¡sa virtuÃ¡lis memÃ³ria hasznÃ¡latÃ¡val. - Egy program hasznÃ¡lhat tÃ¶bb memÃ³riÃ¡t mint a rendelkezÃ©sre Ã¡llÃ³ fizikai mÃ©ret. - Az operÃ¡ciÃ³s rendszer csak a âszÃ¼ksÃ©ges rÃ©sztâ tartja a fizikai memÃ³riÃ¡ban - MMU - virtuÃ¡lis cÃmek fizikai cÃmekre valÃ³ lekÃ©pzÃ©se - MultiprogramozÃ¡s szegmentÃ¡lÃ¡ssal A megoldÃ¡st a virtuÃ¡lismemÃ³ria kezelÃ©s jelentette. Az operÃ¡ciÃ³s rendszer Ãºgy szabadÃt fel memÃ³riÃ¡t az Ã©ppen futÃ³ program szÃ¡mÃ¡ra, hogy a memÃ³riÃ¡ban tÃ¡rolt, de Ã©ppen nem hasznÃ¡lt blokkokat (lapokat) kiÃrja a kÃ¼lsÅ tÃ¡rolÃ³ra, amikor pedig ismÃ©t szÃ¼ksÃ©g van rÃ¡juk, visszaolvassa Åket. Ilyenkor az operÃ¡ciÃ³s rendszer ad a kÃ¶zponti memÃ³riÃ¡bÃ³l egy akkora rÃ©szt, amelyben a folyamat a legfontosabb rÃ©szeit el tudja tÃ¡rolni. A tÃ¶bbit kirakja a hÃ¡ttÃ©rtÃ¡rra (az Ãºn. lapozÃ³fÃ¡jlba, Unix-ban ezt swap-nek hÃvjÃ¡k (a procok akkor is futhatnak ha csak rÃ©szeik vannak a memÃ³riÃ¡ban)). Ez a megoldÃ¡s azÃ©rt mÅ±kÃ¶dik, mert a programok legtÃ¶bbszÃ¶r egy eljÃ¡rÃ¡son belÃ¼l ciklusban dolgoznak, nem csinÃ¡lnak gyakran nagy ugrÃ¡sokat a program egyik vÃ©gÃ©rÅl a mÃ¡sikra. A kÃ¶zponti egysÃ©g fel van szerelve egy Ãºgynevezett memÃ³riakezelÅ egysÃ©ggel (MMU), amely figyeli, hogy olyan kÃ³drÃ©szre kerÃ¼l-e a vezÃ©rlÃ©s, amely nincs benn a kÃ¶zponti memÃ³riÃ¡ban (mert pÃ©ldÃ¡ul a hÃ¡ttÃ©rtÃ¡rra van kirakva). MemÃ³riahasznÃ¡lat szerint a programokat 2 rÃ©szre oszthatjuk: - rezidens (Ã¡llandÃ³an a memÃ³riÃ¡ban van, gyorsabb, tÅ±zfal, vÃrusirtÃ³) - tranziens (csak meghÃvÃ¡skor tÃ¶ltÅdik be, helytakarÃ©kosabb)

ÃllomÃ¡nyrendszerek (file system) A szÃ¡mÃtÃ³gÃ©pek az adatokat kÃ¼lÃ¶nbÃ¶zÅ fizikai hÃ¡ttÃ©rtÃ¡rakon tÃ¡rolhatjÃ¡k, a szÃ¡mÃtÃ³gÃ©p kÃ©nyelmes hasznÃ¡lhatÃ³sÃ¡ga Ã©rdekÃ©ben az operÃ¡ciÃ³s rendszerek egysÃ©ges logikai szemlÃ©letet vezetnek be az adattÃ¡rolÃ¡sra Ã©s adattÃ¡rakra Az operÃ¡ciÃ³s rendszer tÃ¡mogatÃ¡st nyÃºjthat a fÃ¡jl tartalmÃ¡nak kezelÃ©sÃ©ben, a fÃ¡jl szerkezetÃ©nek (adatszerkezet) lÃ©trehozÃ¡sÃ¡ban. ÃllomÃ¡nyrendszer: fÃ¡jlok tÃ¡rolÃ¡sÃ¡nak Ã©s rendszerezÃ©sÃ©nek a mÃ³dszere, ideÃ©rtve a tÃ¡rolt adatokhoz valÃ³ hozzÃ¡fÃ©rÃ©se Ã©s az adatok egyszerÅ± megtalÃ¡lÃ¡sa is

PÃ¡rhuzamossÃ¡ggal kapcsolatos fogalmak, problÃ©mÃ¡k Ã©s megoldÃ¡saik A CPU minden idÅpillanatban egy programot futtat, az egyik program vÃ©grehajtÃ¡sÃ¡rÃ³l a mÃ¡sik program vÃ©grehajtÃ¡sÃ¡ra vÃ¡lt. Ez a pÃ¡rhuzamossÃ¡g illÃºziÃ³jÃ¡t kelti a felhasznÃ¡lÃ³ban, de valÃ³jÃ¡ban nem errÅl van szÃ³. Nem Ã¶sszekeverendÅ a tÃ¶bbprocesszoros rendszerek valÃ³di hardverpÃ¡rhuzamossÃ¡gÃ¡val.

ValÃ³di pÃ¡rhuzamossÃ¡g: - TÃ¶bbprocesszoros rendszerek - AkÃ¡r processzorok szÃ¡zai egy szÃ¡mÃtÃ³gÃ©pben - KÃ¶zÃ¶s sÃn, kÃ¶zÃ¶s Ã³rajel, akÃ¡r kÃ¶zÃ¶s memÃ³ria Ã©s perifÃ©riÃ¡k, gyors kommunikÃ¡ciÃ³ A pÃ¡rhuzamosÃtÃ¡s tipikus megoldÃ¡sa az idÅosztÃ¡s, amikor minden folyamat kap egy-egy Ãºn. idÅszeletet, melynek leteltÃ©t kÃ¶vetÅen egy mÃ¡sik folyamat kapja meg a vezÃ©rlÃ©st. PÃ¡rhuzamossÃ¡g problÃ©mÃ¡i: - a rendszerben futÃ³ folyamatok Ã¡ltalÃ¡ban nem fÃ¼ggetlenek - KÃ¶zÃ¶s erÅforrÃ¡sokat hasznÃ¡lnak - fÃ¼ggÅ folyamatok egyÃ¼ttes viselkedÃ©se Ãºj tÃpusÃº hibÃ¡kat eredmÃ©nyezhet - versenyhelyzetek kialakulÃ¡sa: pÃ¡rhuzamos vÃ©grehajtÃ¡s Ã¡ltal okozott nemdeterminisztikus hibÃ¡s eredmÃ©ny MultiprogramozÃ¡s: Ha az operÃ¡ciÃ³s rendszer egyidÅben tÃ¶bb programot futtat, multiprogramozÃ¡srÃ³l beszÃ©lÃ¼nk, melynek cÃ©lja az erÅforrÃ¡sok jobb kihasznÃ¡lÃ¡sa Ã©s a kÃ©nyelem

16. Processzusok kommunikÃ¡ciÃ³ja, versenyhelyzetek, kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s. Konkurens Ã©s kooperatÃv processzusok. Kritikus szekciÃ³k Ã©s megvalÃ³sÃtÃ¡si mÃ³dszereik: kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s tevÃ©keny vÃ¡rakozÃ¡ssal (megszakÃtÃ¡sok tiltÃ¡sa, vÃ¡ltozÃ³k zÃ¡rolÃ¡sa, szigorÃº vÃ¡ltogatÃ¡s, Peterson megoldÃ¡sa, TSL

utasÃtÃ¡s). AltatÃ¡s Ã©s Ã©bresztÃ©s: termelÅ-fogyasztÃ³ problÃ©ma, szemaforok, mutex-ek, monitorok, Ãzenet, adÃ¡s, vÃ©tel. ÃrÃ³k Ã©s olvasÃ³k problÃ©mÃ¡ja. SorompÃ³k

Processzusok kommunikÃ¡ciÃ³ja, versenyhelyzetek, kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s.

Processzusok kommunikÃ¡ciÃ³ja

- A processzusoknak szÃ¼ksÃ©gÃ¼k vannak a kommunikÃ¡ciÃ³ra
    - Adatok Ã¡tadÃ¡sa az egyik folyamatbÃ³l a mÃ¡siknak (Pipelining)
    - KÃ¶zÃ¶s erÅforrÃ¡sok hasznÃ¡lata (memÃ³ria, nyomtatÃ³, stb.)

Versenyhelyzet

- KooperatÃv processzusok kÃ¶zÃ¶s tÃ¡rolÃ³terÃ¼leten dolgoznak (olvasnak Ã©s Ãrnak).
- Processzusok kÃ¶zÃ¶s adatot olvasnak Ã©s a vÃ©geredmÃ©ny attÃ³l fÃ¼gg, hogy ki Ã©s pontosan mikor fut
- MegoldÃ¡s: Egyszerrecsak egy folyamat lehet kritikus szekciÃ³ban. AmÃg a folyamat kritikus szekciÃ³ban van, azt nem szabad megszakÃtani. EbbÅl a megoldÃ¡sbÃ³l szÃ¡rmazhatnak Ãºj problÃ©mÃ¡k.

KÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s

- Az a mÃ³dszer, ami biztosÃtja, hogy ha egy folyamat hasznÃ¡l valamilyen megosztott, kÃ¶zÃ¶s adatot, akkor mÃ¡s folyamatok ebben az idÅben ne tudjÃ¡k azt elÃ©rni
- pl.: egy adott idÅben csak egy processzus szÃ¡mÃ¡ra engedÃ©lyezett, hogy a nyomtatÃ³nak utasÃtÃ¡sokat kÃ¼ldjÃ¶n
- KÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s miatt elÅfordulhatÃ³ problÃ©mÃ¡k:
    - holtpont (deadlock): processzusok egymÃ¡sra befejezÅdÃ©sÃ©re vÃ¡rnak, hogy a vÃ¡rt erÅforrÃ¡s felszabaduljon
    - Ã©hezÃ©s (starvation): egy processzusnak hatÃ¡rozatlan ideig vÃ¡rnia kell egy erÅforrÃ¡s hasznÃ¡latÃ¡ra

Kritikus szekciÃ³

- A program azon rÃ©sze, amelyben a programunk a kÃ¶zÃ¶s adatokat hasznÃ¡lja
- SzabÃ¡lyok:
    - legfeljebb egy proc lehet kritikus szekciÃ³jÃ¡ban
    - kritikus szekciÃ³n kivÃ¼li proc nem befolyÃ¡solhatja mÃ¡sik proc kritikus szekciÃ³ba lÃ©pÃ©sÃ©t
    - vÃ©ges idÅn belÃ¼l bÃ¡rmely kritikus szekciÃ¼ba lÃ©pni kivÃ¡nÃ³ proc belÃ©phet

Kritikus szekciÃ³k Ã©s megvalÃ³sÃtÃ¡si mÃ³dszereik: kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s tevÃ©keny vÃ¡rakozÃ¡ssal (megszakÃtÃ¡sok tiltÃ¡sa, vÃ¡ltozÃ³k zÃ¡rolÃ¡sa, szigorÃº vÃ¡ltogatÃ¡s, Peterson megoldÃ¡sa, TSL utasÃtÃ¡s).

LÃ¡thattuk, hogy a kritikus szekciÃ³ba valÃ³ belÃ©pÃ©s nem feltÃ©tel nÃ©lkÃ¼li. Hogyan biztosÃthatjuk a kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s teljesÃ¼lÃ©sÃ©t? - Hardware-es mÃ³dszer - MegszakÃtÃ¡sok tiltÃ¡sÃ¡val - letiltjuk a megszakÃtÃ¡st a kritikus szekciÃ³ba lÃ©pÃ©s utÃ¡n, majd Ãºjra engedÃ©lyezzÃ¼k, mielÅtt elhagyja azt, Ãgy nem fordulhat elÅ Ã³ramegszakÃtÃ¡s, azaz a CPU nem fog mÃ¡sik processzusra vÃ¡ltani - jÃ³l hasznÃ¡lhatÃ³, de Ã¡ltalÃ¡nosan nem biztos, hogy a legszerencsÃ©sebb - a legegyszerÅ±bb hiba, hogy elfelejtjÃ¼k Ãºjra engedÃ©lyezni a megszakÃtÃ¡st a kritikus szekciÃ³ vÃ©gÃ©n - TSL utasÃtÃ¡s segÃtsÃ©gÃ©vel - A mai rendszerekben a processzornak van egy âTSL reg, lockâ (TSL EAX, lock) formÃ¡jÃº utasÃtÃ¡sa (TSL â Test and Set Lock). - Ez az utasÃtÃ¡s beolvassa a LOCK memÃ³riaszÃ³ tartalmÃ¡t a âregâ regiszterbe, majd egy nem nulla Ã©rtÃ©ket Ãr a âlockâ memÃ³riacÃmre. - A CPU zÃ¡rolja a memÃ³riasÃnt, azaz tiltva van a memÃ³ria elÃ©rÃ©s a CPU-knak a mÅ±velet befejezÃ©sÃ©ig. - A mÅ±velet befejezÃ©sekor 0 Ã©rtÃ©k kerÃ¼l a LOCK memÃ³riaterÃ¼letre - Szoftware-es mÃ³dszer - SzigorÃº vÃ¡ltogatÃ¡s mÃ³dszere A folyamat B folyamat - - Peterson-fÃ©le megoldÃ¡s - Van kÃ©t metÃ³dus a kritikus szekciÃ³ba valÃ³ belÃ©pÃ©sre (enter_region) Ã©s kilÃ©pÃ©sre (leave_region). A kritikus szekciÃ³ba lÃ©pÃ©s elÅtt a processzus meghÃvja az enter_region eljÃ¡rÃ¡st, kilÃ©pÃ©skor pedig a leave_region eljÃ¡rÃ¡st. Az enter_region eljÃ¡rÃ¡s biztosÃtani fogja, hogy a mÃ¡sik processzus vÃ¡rakozik, ha szÃ¼ksÃ©ges. - - VÃ¡ltozÃ³k zÃ¡rolÃ¡sa - Van egy osztott zÃ¡rolÃ¡si vÃ¡ltozÃ³, aminek a kezdeti Ã©rtÃ©ke 0. Kritikus szekciÃ³ba lÃ©pÃ©s elÅtt a processzus teszteli ezt a vÃ¡ltozÃ³t. Ha 0 az Ã©rtÃ©ke, akkor 1-re Ã¡llÃtja Ã©s belÃ©p a kritikus szekciÃ³ba. Ha az Ã©rtÃ©ke 1, akkor vÃ¡rakozik, amÃg nem lesz 0.

AltatÃ¡s Ã©s Ã©bresztÃ©s: termelÅ-fogyasztÃ³ problÃ©ma, szemaforok, mutex-ek, monitorok, Ãzenet, adÃ¡s, vÃ©tel.

AltatÃ¡s-Ã©bresztÃ©s

Ahogy lÃ¡ttuk az elÅzÅ, tevÃ©keny vÃ¡rakozÃ¡st hasznÃ¡lÃ³ versenyhelyzet-elkerÃ¼lÅ megoldÃ¡sokban a legfontosabb gond az, hogy processzoridÅt pazarolnak. Ahhoz, hogy a drÃ¡ga processzoridÅt se pazaroljuk, olyan megoldÃ¡st lehet javasolni, ami vagy blokkolni tud egy folyamatot (aludni kÃ¼ldi), vagy fel tudja Ã©breszteni ebbÅl a blokkolt Ã¡llapotbÃ³l.

A tevÃ©keny vÃ¡rakozÃ¡s feloldÃ¡sÃ¡ra az egyik eszkÃ¶z a sleep-wakeup rendszerhÃvÃ¡s pÃ¡ros. A lÃ©nyege, hogy a sleep rendszerhÃvÃ¡s blokkolja a hÃvÃ³t, azaz fel lesz fÃ¼ggesztve, amÃg egy mÃ¡sik processzus fel nem Ã©breszti. A wakeup rendszerhÃvÃ¡s a paramÃ©terÃ¼l kap egy processzus azonosÃtÃ³t, amely segÃtsÃ©gÃ©vel felÃ©breszti az adott processzust, tehÃ¡t nem lesz blokkolva tovÃ¡bbÃ¡.

TermelÅ-fogyasztÃ³ problÃ©ma

VÃ©ges mÃ©retÅ± memÃ³riaterÃ¼leten (tÃ¡rolÃ³n) dolgozik kÃ©t processzus (osztoznak). A gyÃ¡rtÃ³ adatokat helyez el a tÃ¡rolÃ³n, a fogyasztÃ³ kiveszi az adatokat a tÃ¡rolÃ³bÃ³l Ã©s feldolgozza azokat, viszont a memÃ³ria vÃ©ges. Ha a tÃ¡rolÃ³ tele van Ã©s a gyÃ¡rtÃ³ elemet akar berakni, akkor elalszik, majd felÃ©breszti a fogyasztÃ³, ha egy elemet kivesz a tÃ¡rolÃ³bÃ³l. FordÃtva is: ha a tÃ¡rolÃ³ Ã¼res, Ã©s a fogyasztÃ³ ki akar venni egy elemet, akkor elalszik, Ã©s majd felÃ©breszti a gyÃ¡rtÃ³, ha legyÃ¡rtott egy eleme

Szemafor

- âA vonat megÃ¡ll egy piros szemafor elÅtt, Ã©s addig vÃ¡rakozik, amÃg szabad utat nem kap, mert valamilyen oknÃ¡l fogva (elaludt a bakter, foglalt a pÃ¡lya stb.) a tovÃ¡bbhaladÃ¡s meg van tiltva.â
- A szemafor a szÃ¡mÃtÃ³gÃ©p-programozÃ¡sban hasznÃ¡lt vÃ¡ltozÃ³ vagy absztrakt adattÃpus, amit az osztott erÅforrÃ¡sokhoz valÃ³ hozzÃ¡fÃ©rÃ©sek szabÃ¡lyozÃ¡sÃ¡hoz hasznÃ¡lnak a tÃ¶bbszÃ¡lÃº kÃ¶rnyezetekben.
- Ha Ã©rtÃ©ke pozitÃv, akkor nyitott Ã¡llapotban van, ha nulla, akkor tilosat mutat
- Amekkora Ã©rtÃ©kkel inicializÃ¡ljuk a szemafort annyi âvonatotâ enged Ã¡t, mielÅtt tilosat mutatna.
- pl.: TekintsÃ¼nk egy egÃ©sz szÃ¡mot. Legyen, mondjuk, a kezdÅÃ©rtÃ©ke egy. Amikor a kritikus mÅ±velethez Ã©rek, akkor azt mondom, hogy jelzem az erÅforrÃ¡s-hasznÃ¡lati igÃ©nyemet. A jelzÃ©s jelentse azt, hogy eggyel csÃ¶kkentem a szÃ¡m Ã©rtÃ©kÃ©t. Ezt szokÃ¡s âdownâ vagy sok mÃ¡s helyen âPâ operÃ¡ciÃ³nak is nevezni. Ha a csÃ¶kkentÃ©s eredmÃ©ny nem negatÃv lesz, akkor szabad az Ãºt, Ã©s vÃ©gzi a dolgÃ¡t a program. Ha ezutÃ¡n Ã©rkezik egy mÃ¡sik folyamat, ami ugyanezt az erÅforrÃ¡st szeretnÃ© hasznÃ¡lni, szintÃ©n hasonlÃ³ mÃ³don kezdi a dolgot, de neki mÃ¡r a P operÃ¡ciÃ³ pirosra Ã¡llÃtja a szemafort hiszen az âegÃ©szâ Ã©rtÃ©ke mÃnusz egy lesz. Ekkor ez a mÃ¡sodik folyamat mindaddig vÃ¡r, amÃg a szemafor Ã©rtÃ©kÃ©t egy Ãºgynevezett âupâ vagy âVâ operÃ¡ciÃ³val â ami az eggyel valÃ³ nÃ¶velÃ©st jelenti â fel nem szabadÃtja az erÅforrÃ¡st ami utÃ¡n a P operÃ¡ciÃ³nÃ¡l vÃ¡rakozÃ³ program tovÃ¡bb haladhat. Ezzel tulajdonkÃ©ppen Ãºjra tilos jelzÃ©s lesz Ã©rvÃ©nyben a kritikus erÅforrÃ¡sra a kezdÅÃ©rtÃ©k egy volt.

Mutex

- Olyan speciÃ¡lis szemafor, amelynek csak kÃ©t Ã©rtÃ©ke lehet
- Ha csak kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s biztosÃtÃ¡sÃ¡ra kell a szemafort lÃ©trehozni, Ã©s nincs szÃ¼ksÃ©g annak szÃ¡mlÃ¡lÃ¡si kÃ©pessÃ©gÃ©re, akkor azt egy kezdÅÃ©rtÃ©kkel hozzuk lÃ©tre. Ezt a kÃ©tÃ¡llapotÃº (Ã©rtÃ©ke 0 Ã©s 1) szemafort sok kÃ¶rnyezetben speciÃ¡lis nÃ©vvel, az angol kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s kifejezÃ©sbÅl mutexnek nevezzÃ¼k.
- Ha egy folyamatnak zÃ¡rolÃ¡sra van szÃ¼ksÃ©ge, a âmutex_lockâ eljÃ¡rÃ¡st hÃvja, mÃg ha a zÃ¡rolÃ¡st meg akarja szÃ¼ntetni, a âmutex_unlockâ utasÃtÃ¡st hÃvja.
- Aki mÃ¡sodszor (vagy harmadszor) hÃvja a âmutex_lockâ eljÃ¡rÃ¡st, az blokkolÃ³dik, Ã©s csak a âmutex_unlockâ hatÃ¡sÃ¡ra tudja folytatni a vÃ©grehajtÃ¡st.

Monitor

- EljÃ¡rÃ¡sok, vÃ¡ltozÃ³k Ã¡s adatszerkezetek egyÃ¼ttese egy speciÃ¡lis modulba Ã¶sszegyÅ±jtve, hogy hasznÃ¡lhatÃ³ legyen a kÃ¶lcsÃ¶nÃ¶s kizÃ¡rÃ¡s megvalÃ³sÃtÃ¡sÃ¡ra
- Legfontosabb tulajdonsÃ¡ga, hogy egy adott idÅpillanatban csak egy proc lehet aktÃv benne
- A processzusok bÃ¡rmikor hÃvhatjÃ¡k a monitorban lÃ©vÅ eljÃ¡rÃ¡sokat, de nem Ã©rhetik el a belsÅ adatszerkezeteit (mint OOP-nÃ¡l)
- wait(c): alvÃ³ Ã¡llapotba kerÃ¼l a vÃ©grehajtÃ³ proc
- signal(c): a c miatt alvÃ³ procot felÃ©breszti

Ãzenet, adÃ¡s, vÃ©tel.

- Folyamatok egyÃ¼ttmÅ±kÃ¶dÃ©shez informÃ¡ciÃ³ cserÃ©re van szÃ¼ksÃ©g. KÃ©t mÃ³d:
  - kÃ¶zÃ¶s tÃ¡rterÃ¼leten keresztÃ¼l
  - kommunikÃ¡ciÃ³s csatornÃ¡n keresztÃ¼l (egy vagy kÃ©tirÃ¡nyÃº)
- Folyamat fommunikÃ¡ciÃ³ fajtÃ¡k:
  - KÃ¶zvetlen kÃ¶mmunikÃ¡ciÃ³

- csak egy csatorna lÃ©tezik, Ã©s mÃ¡s folyamatok nem hasznÃ¡lhatjÃ¡k

  - KÃ¶zvetett kommunikÃ¡ciÃ³

- KÃ¶zbÃ¼lsÅ adatszerkezeten (pl. postalÃ¡dÃ¡n (mailbox)) keresztÃ¼l

valÃ³sul meg.

- Aszimmetrikus
  - AdÃ³ vagy vevÅ megnevezi, hogy melyik folyamattal akar kommunikÃ¡lni
  - A mÃ¡sik fÃ©l egy kaput (port) hasznÃ¡l, ezen keresztÃ¼l tÃ¶bb folyamathoz, is kapcsolÃ³dhat.
  - Tipikus eset: a vevÅhÃ¶z tartozik a kapu, az adÃ³knak kell a vevÅ folyamatot Ã©s annak a kapujÃ¡t megnevezni. (Pl. szerver, szolgÃ¡ltatÃ³ folyamat)
  -
- Ãzenetszã³rÃ¡s

- A kÃ¶zeg tÃ¶bb folyamatot kÃ¶t Ã¶ssze.

- MÅ±veletek:
  - send(cÃ©l, &Ã¼zenet)
  - receive(forrÃ¡s, &Ã¼zenet)

ÃrÃ³k Ã©s olvasÃ³k problÃ©mÃ¡ja. SorompÃ³k.

ÃrÃ³k Ã©s olvasÃ³k problÃ©mÃ¡ja

TÃ¶bb proc egymÃ¡ssal versengve Ãrja Ã©s olvassa ugyanazt az adatot. Megengedett az egyidejÅ± olvasÃ¡s, de ha egy proc Ãrni akar, akkor mÃ¡s procok sem nem Ãrhatnak se nem olvashatnak. (pl, adatbÃ¡zisok, fÃ¡jlok, hÃ¡lÃ³zat)

SorompÃ³k: - SorompÃ³ primitÃv - KÃ¶nyvtÃ¡ri eljÃ¡rÃ¡s - FÃ¡zisokra osztjuk az alkalmazÃ¡st - SzabÃ¡ly - Egyetlen processzus sem mehet tovÃ¡bb a kÃ¶vetkezÅ fÃ¡zisra, amÃg az Ã¶sszes processzus nem Ã¡ll kÃ©szen - SorompÃ³ elhelyezÃ©se mindegyik fÃ¡zis vÃ©gÃ©re - Amikor egy processzus a sorompÃ³hoz Ã©r, akkor addig blokkolÃ³dik ameddig az Ã¶sszes processzus el nem Ã©ri a sorompÃ³t - A sorompÃ³ az utolsÃ³ processzus beÃ©rkezÃ©se utÃ¡n elengedi a azokat - Nagy mÃ¡trix-okon vÃ©gzett pÃ¡rhuzamos mÅ±veletek

1. AdatbÃ¡zis-tervezÃ©s: A relÃ¡ciÃ³s adatmodell fogalma. Az egyed-kapcsolat diagram Ã©s lekÃ©pezÃ©se relÃ¡ciÃ³s modellre, kulcsok fajtÃ¡i. FunkcionÃ¡lis fÃ¼ggÅsÃ©g, a normalizÃ¡lÃ¡s cÃ©lja, normÃ¡lformÃ¡k

1. AdatbÃ¡zis-tervezÃ©s: A relÃ¡ciÃ³s adatmodell fogalma. Az egyed-kapcsolat diagram Ã©s lekÃ©pezÃ©se relÃ¡ciÃ³s modellre, kulcsok fajtÃ¡i. FunkcionÃ¡lis fÃ¼ggÅsÃ©g, a normalizÃ¡lÃ¡s cÃ©lja, normÃ¡lformÃ¡k

A relÃ¡ciÃ³s adatmodell fogalma

A relÃ¡ciÃ³s adatmodell mind az adatokat, mind a kÃ¶ztÃ¼k lÃ©vÅ kapcsolatokat kÃ©tdimenziÃ³s tÃ¡blÃ¡kban tÃ¡rolja.

AttribÃºtum:

- nÃ©vvel, Ã©rtÃ©ktartomÃ¡nnyal megadott tulajdonsÃ¡g
- Z Ã©rtÃ©ktartomÃ¡nyÃ¡t dom(Z) jelÃ¶li
- csak elemi tÃpusÃº Ã©rtÃ©kekbÅl Ã¡llhat
- gyakran megadjuk az Ã¡brÃ¡zolÃ¡s hosszÃ¡t is

RelÃ¡ciÃ³sÃ©ma:

- nÃ©vvel ellÃ¡tott attribÃºtumhalmaz
- R(A), ahol A az attribÃºtumok halmaza
- nÃ©vÃ¼tkÃ¶zÃ©s esetÃ©n kiÃrhatjuk a tÃ¡bla nevÃ©t is az attribÃºtum elÃ©

A relÃ¡ciÃ³sÃ©ma nem tÃ¡rol adatot! Csak szerkezeti leÃrÃ¡st jelent.

Az adatok relÃ¡ciÃ³kkal adhatÃ³k meg. Egy R(A) sÃ©ma feletti relÃ¡ciÃ³ A Ã©rtÃ©ktartomÃ¡nyainak direktszorzatÃ¡nak egy rÃ©szhalmaza (mindegyik attribÃºtum Ã©rtÃ©kei kÃ¶zÃ¼l vÃ¡lasztunk egyet, Ã©s ezt egy vektorba pakoljuk). Egy ilyen relÃ¡ciÃ³ mÃ¡r megjelenÃthetÅ adattÃ¡bla formÃ¡jÃ¡ban, egy relÃ¡ciÃ³ a tÃ¡blÃ¡zat egy sorÃ¡nak felel meg.

Az egyed-kapcsolat diagram Ã©s lekÃ©pezÃ©se relÃ¡ciÃ³s modellre

EK-diagram

Az egyed-kapcsolat modell konkrÃ©t adatmodelltÅl fÃ¼ggetlenÃ¼l, szemlÃ©letesen adja meg az adatbÃ¡zis szerkezetÃ©t.

Egyed vagy entitÃ¡s

- a valÃ³s vilÃ¡g egy objektuma
- szeretnÃ©nk rÃ³la informÃ¡ciÃ³t tÃ¡rolni az adatbÃ¡zisban
- egyedtÃpus: Ã¡ltalÃ¡nossÃ¡gban jelent egy valÃ³s objektumot
- egyedpÃ©ldÃ¡ny: egy konkrÃ©t objektum
- gyenge egyed: ha az egyedet nem hatÃ¡rozza meg egyÃ©rtelmÅ±en attribÃºtumainak semmilyen rÃ©szhalmaza

TulajdonsÃ¡g vagy attribÃºtum

- az egyed egy jellemzÅje
- tulajdonsÃ¡gtÃpus vs tulajdonsÃ¡gpÃ©ldÃ¡ny
- az attribÃºtumok egy olyan legszÅ±kebb rÃ©szhalmazÃ¡t, amely egyÃ©rtelmÅ±en meghatÃ¡rozza az egyedet, kulcsnak nevezzÃ¼k

Kapcsolatok

- egyedek kÃ¶zÃ¶tt alakulhatnak ki
- kapcsolattÃpus: pl felhasznÃ¡lÃ³ Ã©s Ã¼zenet kÃ¶zÃ¶tt
- kapcsolatpÃ©ldÃ¡ny: pl Kis JÃ³zsef Ã©s a 69420. Ã¼zenet
- kapcsolatoknak is lehet tulajdonsÃ¡ga

Azt a modellt, amelyben az adatbÃ¡zis a tÃ¡rolandÃ³ adatokat egyedekkel, tulajdonsÃ¡gokkal Ã©s kapcsolatokkal Ãrja le, egyed-kapcsolat modellnek nevezzÃ¼k, a hozzÃ¡ kapcsolÃ³dÃ³ diagramot pedig egyed-kapcsolat diagrammnak.

A diagramon

- az egyedeket tÃ©glalappal
- a tulajdonsÃ¡gokat ellipszissel
- a kulcsot alÃ¡hÃºzÃ¡ssal
- a kapcsolatokat rombusszal

jelÃ¶ljÃ¼k.

EK lekÃ©pezÃ©se relÃ¡ciÃ³s adatmodellre

Egyedek lekÃ©pezÃ©se

- minden egyedhez egy relÃ¡ciÃ³sÃ©mÃ¡t Ãrunk fel, melynek neve az egyed neve, attribÃºtumai pedig az egyed attribÃºtumai, kulcsa pedig az egyed kulcsa
- gyenge egyednÃ©l az attribÃºtumokhoz hozzÃ¡ kell venni a meghatÃ¡rozÃ³ kapcsolatokon keresztÃ¼k csatlakozÃ³ egyedek kulcsattribÃºtumait is, kÃ¼lsÅ kulcskÃ©nt

Ãsszetett attr. lekÃ©pezÃ©se

- Ã¶sszetett attribÃºtumot helyettesÃtÃ¼nk az Åt alkotÃ³ elemi attribÃºtumokkal

TÃ¶bbÃ©rtÃ©kÅ± attribÃºtumok lekÃ©pezÃ©se

- egyik lehetÅsÃ©g:
  - eltekintÃ¼nk attÃ³l, hogy tÃ¶bbÃ©rtÃ©kÅ±, Ã©s egyszerÅ± szÃ¶vegkÃ©nt tÃ¡roljuk
  - hÃ¡trÃ¡nya, hogy nem kezelhetÅk kÃ¼lÃ¶n kÃ¼lÃ¶n az elemek
- mÃ¡sik lehetÅsÃ©g:
  - minden sorbÃ³l annyit veszÃ¼nk fel, ahÃ¡ny Ã©rtÃ©ke van a tÃ¶bbÃ©rtÃ©kÅ± attribÃºtumnak
  - hÃ¡trÃ¡nya a sok fÃ¶lÃ¶sleges sor
  - kulcsok elromlanak
  - kerÃ¼lendÅ
- harmadik lehetÅsÃ©g
  - Ãºj tÃ¡blÃ¡t veszÃ¼nk fel, ahova kigyÅ±jtjÃ¼k, hogy melyik sorhoz milyen Ã©rtÃ©kei tartoznak a tÃ¶bbÃ©rtÃ©kÅ± attribÃºtumnak

- akÃ¡r kÃ¼lÃ¶n kigyÅ±jthetjÃ¼k egy tÃ¡blÃ¡ba az Ã¶sszes lehetsÃ©ges Ã©rtÃ©kÃ©t a tÃ¶bbÃ©rtÃ©kÅ± attribÃºtumnak, Ã©s egy kapcsolÃ³tÃ¡blÃ¡val kÃ¶tjÃ¼k Ã¶ssze az egyeddel

Kapcsolatok lekÃ©pezÃ©se

- minden kapcsolathoz felveszÃ¼nk egy Ãºj sÃ©mÃ¡t
- neve a kapcsolat neve, attribÃºtumai a kapcsolÃ³dÃ³ egyedek kulcsattribÃºtumai Ã©s a kapcsolat sajÃ¡t attribÃºtumai
- meg kell hatÃ¡rozni ennek a sÃ©mÃ¡nak is a kulcsÃ¡t
- ha ez a kulcs megegyezik valamelyik kapcsolt egyed kulcsÃ¡val, akkor ez a sÃ©ma beolvaszthatÃ³ abba az egyedbe, ezt hÃvjuk konszolidÃ¡ciÃ³nak, ez a gyakorlatban egy lÃ©pÃ©sben is elvÃ©gezhetÅ persze
- 1:1 kapcsolat esetÃ©n az egyik tetszÅlegesen vÃ¡lasztott egyedbe beolvaszthatjuk a kapcsolat sÃ©mÃ¡jÃ¡t
- 1:N kapcsolat esetÃ©n az N oldali egyedet bÅvÃtjÃ¼k a mÃ¡sik egyed kulcsattribÃºtumaival, Ã©s a kapcsolat sajÃ¡t attribÃºtumaival
- N:M kapcsolat esetÃ©n Ãºj sÃ©mÃ¡t veszÃ¼nk fel

SpecializÃ¡lÃ³ kapcsolatok lekÃ©pezÃ©se

Minden megkÃ¶zelÃtÃ©snek lehetnek hÃ¡trÃ¡nyai, mÃ©rlegelnÃ¼nk kell

ElsÅ lehetÅsÃ©g

- fÅtÃpus Ã©s altÃpus is kÃ¼lÃ¶n sÃ©mÃ¡ban, Ã©s az altÃpus attribÃºtumai kÃ¶zÃ© felvesszÃ¼k a fÅtÃpus attribÃºtumait is
- minden egyedpÃ©ldÃ¡ny csak egy tÃ¡blÃ¡ban fog szerepelni

MÃ¡sodik lehetÅsÃ©g

- minden altÃpushoz Ãºj sÃ©ma, de abban csak a fÅtÃpus kulcsattribÃºtumai jelennek meg
- minden egyedpÃ©ldÃ¡ny szerepel a sajÃ¡t altÃpusÃ¡nak tÃ¡blÃ¡jÃ¡ban Ã©s a fÅtÃpus tÃ¡blÃ¡jÃ¡ban is

Harmadik lehetÅsÃ©g

- egy kÃ¶zÃ¶s tÃ¡bla az Ã¶sszes lehetsÃ©ges attribÃºtummal
- minden sorban csak a relevÃ¡ns cellÃ¡kat tÃ¶ltjÃ¼k ki

Kulcsok fajtÃ¡i

Szuperkulcs

- egyÃ©rtelmÅ±en azonosÃtja a tÃ¡bla sorait
- R(A) bÃ¡rmely kÃ©t sora kÃ¼lÃ¶nbÃ¶zik a szuperkulcson
- mivel a tÃ¡blÃ¡ban Ã¡ltalÃ¡ban nem engedÃ¼nk meg ismÃ©tlÅdÅ sorokat, ezÃ©rt ha az Ã¶sszes attribÃºtumot vesszÃ¼k, az midnig szuperkulcs

Kulcs

- olyan szuperkulcs, amelynek egyetlen valÃ³di rÃ©szhalmaza sem szuperkulcs
- ha egyelemÅ±, egyszerÅ± kulcsnak nevezzÃ¼k
- ha tÃ¶bbelemÅ±, Ã¶sszetettnek
- elÅfordulhat, hogy van tÃ¶bb kulcs is, ekkor kivÃ¡lasztunk egyet
- a kivÃ¡laszott kulcsot elsÅdleges kulcsnak nevezzÃ¼k

KÃ¼lsÅ kulcs

- mÃ¡sik, vagy ugyanazon sÃ©ma elsÅdleges kulcsÃ¡ra vonatkozik

Mind az elsÅdleges kulcs Ã©s a kÃ¼lsÅ kulcsok is a sÃ©mÃ¡ra vonatkozÃ³ feltÃ©telek, fÃ¼ggetlenek az adatoktÃ³l

FunkcionÃ¡lis fÃ¼ggÅsÃ©g

P Ã©s Q attribÃºtumhalmazok, az R(A) sÃ©mÃ¡n P-tÅl funkcionÃ¡lisan fÃ¼gg Q, ha bÃ¡rmilyen R feletti tÃ¡bla esetÃ©n ha P-n megegyezik kÃ©t sor, akkor Q-n is meg fog egyezni.

TriviÃ¡lis, ha Q rÃ©szhalmaza P-nek, Ã©s nemtriviÃ¡lis, ha P-nek Ã©s Q-nak nincs kÃ¶zÃ¶s attribÃºtuma.

Pl a felhasznÃ¡lÃ³nÃ©vtÅl funkcionÃ¡lisan fÃ¼gg az email sokszor.

NormalizÃ¡lÃ¡s cÃ©lja, normÃ¡lformÃ¡k

TÃ¡rolhatnÃ¡nk az Ã¶sszes adatunkat egy nagy tÃ¡blÃ¡ban is, de ilyenkor gondok merÃ¼lhetnek fel az adatbÃ¡zismÅ±veletek sorÃ¡n, illetve nagyon redundÃ¡ns lenne az adattÃ¡rolÃ¡s. A normalizÃ¡lÃ¡s cÃ©lja kisebb tÃ¡blÃ¡k lÃ©trehozÃ¡sa a redundancia elkerÃ¼lÃ©se Ã©rdekÃ©ben.

NormÃ¡lformÃ¡k

DekompozÃciÃ³ segÃtsÃ©gÃ©vel megszÃ¼ntetjÃ¼k lÃ©pÃ©srÅl lÃ©pÃ©sre a redundanciÃ¡t Ãºgy, hogy a sÃ©mÃ¡ban lÃ©vÅ fÃ¼ggÅsÃ©gekre egyre szigorÃºbb feltÃ©teleket adunk.

ElsÅdleges, mÃ¡sodlagos attribÃºtum: szerepel a sÃ©ma valamelyik kulcsÃ¡ban, ha nem akkor mÃ¡sodlagos TranzitÃv, kÃ¶zvetlen fÃ¼ggÃ©s: Ha X-tÅl fÃ¼gg Z, Ã©s van olyan Y, hogy X -> Y Ã©s Y -> Z, ellenkezÅ esetben kÃ¶zvetlenÃ¼l fÃ¼gg

1NF:

- Ha az attribÃºtumok Ã©rtÃ©ktartomÃ¡nya csak egyszerÅ± adatokbÃ³l Ã¡ll (nincs tÃ¶bbszÃ¶rÃ¶s vagy Ã¶sszetett attribÃºtum)

2NF:

- Ha minden mÃ¡sodlagos attribÃºtum teljesen fÃ¼gg bÃ¡rmely kulcstÃ³l

3NF:

- Minden mÃ¡sodlagos attribÃºtum kÃ¶zvetlenÃ¼l fÃ¼gg bÃ¡rmely kulcstÃ³l, azaz nincs tranzitÃv fÃ¼ggÃ©s

BCNF:

- Egy relÃ¡ciÃ³sÃ©ma Boyce-Codd normÃ¡lformÃ¡ban van, ha bÃ¡rmely nemtriviÃ¡lis L -> B fÃ¼ggÃ©s esetÃ©n L szuperkulcs.

6ef0bd001a5ea93950836f58b711eb2c3bb3daee

2. Az SQL adatbÃ¡zisnyelv: Az adatdefinÃciÃ³s nyelv (DDL) Ã©s az adatmanipulÃ¡ciÃ³s nyelv (DML). RelÃ¡ciÃ³sÃ©mÃ¡k definiÃ¡lÃ¡sa, megszorÃtÃ¡sok tÃpusai Ã©s lÃ©trehozÃ¡suk. AdatmanipulÃ¡ciÃ³s lehetÅsÃ©gek Ã©s lekÃ©rdezÃ©sek

SQL

Structured Query Language

Arra szolgÃ¡l, hogy adatokat kezeljÃ¼nk vele

- beszÃºrÃ¡s
- tÃ¶rlÃ©s
- mÃ³dosÃtÃ¡s
- lekÃ©rdezÃ©s

A nyelv elemeit kÃ©t fÅ rÃ©szre oszthatjuk.

Az adatdefinÃciÃ³s nyelv

Ide tartoznak az adatbÃ¡zisok, sÃ©mÃ¡k, tÃpusok definÃciÃ³s utasÃtÃ¡sai, pl:

- CREATE DATABASE
- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE TRIGGER

Az adat manipulÃ¡ciÃ³s nyelv

Ide tartoznak a beszÃºrÃ³, mÃ³dosÃtÃ³, tÃ¶rlÅ, lekÃ©rdezÅ utasÃtÃ¡sok.

- INSERT INTO
- UPDATE
- DELETE FROM

- SELECT

Egyes irodalmak kÃ¼lÃ¶nvÃ¡lasztjÃ¡k a lekÃ©rdezÅ utasÃtÃ¡sokat a manipulÃ¡ciÃ³s utasÃtÃ¡soktÃ³l.

RelÃ¡ciÃ³sÃ©mÃ¡k definiÃ¡lÃ¡sa, megszorÃtÃ¡sok tÃpusai Ã©s lÃ©trehozÃ¡suk

RelÃ¡ciÃ³sÃ©mÃ¡kat a CREATE TABLE utasÃtÃ¡ssal hozhatunk lÃ©re. A sÃ©mÃ¡k kÃ¼lÃ¶nbÃ¶znek a tÃ¡blÃ¡ktÃ³l, Ã©s nevÃ©vel ellentÃ©tben a CREATE TABLE utasÃtÃ¡s csak a relÃ¡ciÃ³sÃ©mÃ¡t hozza lÃ©tre. A tÃ¡bla mÃ¡r az adatrekordok halmazÃ¡t jelenti.

MegszorÃtÃ¡sok

OszlopfeltÃ©telek:

Csak az adott mezÅre vonatkoznak

- PRIMARY KEY, az elsÅdleges kulcs
- UNIQUE, kulcs, minden Ã©rtÃ©k egyszer fordulhat elÅ az oszlopban
- NOT NULL, az oszlop Ã©rtÃ©ke nem lehet NULL, azaz kÃ¶telezÅ kitÃ¶lteni
- REFERENCES T(oszlop), a T tÃ¡bla oszlop oszlopÃ¡ra vonatkozÃ³ kÃ¼lsÅ kulcs
- DEFAULT tartalom, az oszlop alapÃ©rtelmezett Ã©rtÃ©ke tartalom lesz

TÃ¡blafeltÃ©telek

Ha tÃ¶bb oszlopra is vonatkoznak feltÃ©telek, azt itt tudjuk megadni.

- PRIMARY KEY(oszloplista), az elsÅdleges kulcs
- UNIQUE (oszloplista), kulcs, minden Ã©rtÃ©k egyszer fordulhat elÅ az oszlopban
- FOREIGN KEY (oszloplista) REFERENCES T(oszloplista), a T tÃ¡bla oszloplista oszloplistÃ¡jÃ¡ra vonatkozÃ³ kÃ¼lsÅ kulcs

KÃ¼lsÅ kulcs feltÃ©telek Ã©s szabÃ¡lyok

Az integritÃ¡s megÅrzÃ©se szempontjÃ¡bÃ³l a kÃ¼lsÅ kulcsokhoz meghatÃ¡rozhatjuk azt is, hogy hogyan viselkedjenek a hivatkozott kulcs tÃ¶rlÃ©se vagy mÃ³dosÃtÃ¡sa esetÃ©n.

ON DELETE

- RESTRICT, ha van a tÃ¶rlendÅ rekord kulcsÃ¡ra van vonatkozÃ³ kÃ¼lsÅ kulcs, megtiltjuk a tÃ¶rlÃ©st
- SET NULL, a tÃ¶rlendÅ rekord kulcsÃ¡ra hivatkozÃ³ kÃ¼lsÅ kulcs Ã©rtÃ©kÃ©t NULL-ra Ã¡llÃtjuk
- NO ACTION, a tÃ¶rlendÅ rekord kulcsÃ¡ra vonatkozÃ³ kÃ¼lsÅ kulcs Ã©rtÃ©ke nem vÃ¡ltozik
- CASCADE, a tÃ¶rlendÅ rekord kulcsÃ¡ra hivatkozÃ³ kÃ¼lsÅ kulcsÃº rekordok is tÃ¶rlÅdnek

ON UPDATE

- RESTRICT, ha van a mÃ³dosÃtandÃ³ rekord kulcsÃ¡ra van vonatkozÃ³ kÃ¼lsÅ kulcs, megtiltjuk a mÃ³dosÃtÃ¡st
- SET NULL, a mÃ³dosÃtandÃ³ rekord kulcsÃ¡ra hivatkozÃ³ kÃ¼lsÅ kulcs Ã©rtÃ©kÃ©t NULL-ra Ã¡llÃtjuk
- NO ACTION, a mÃ³dosÃtandÃ³ rekord kulcsÃ¡ra vonatkozÃ³ kÃ¼lsÅ kulcs Ã©rtÃ©ke nem vÃ¡ltozik
- CASCADE, a mÃ³dosÃtandÃ³ rekord kulcsÃ¡ra hivatkozÃ³ kÃ¼lsÅ kulcsÃº rekordok is az Ãºj Ã©rtÃ©kre vÃ¡ltoznak

TÃ¡blÃ¡kra Ã©s attribÃºtumokra vonatkozÃ³ megszorÃtÃ¡sok

ElsÅdleges feladata, hogy megelÅzzÃ¼k az adatbeviteli hibÃ¡kat, Ã©s elkerÃ¼ljÃ¼k a hiÃ¡nyzÃ³ adatokat a kÃ¶telezÅ mezÅkbÅl.

NOT NULL: a cella Ã©rtÃ©kÃ©t kÃ¶telezÅ kitÃ¶lteni, nem lehet NULL

CHECK (feltÃ©tel): ellenÅrzÅ feltÃ©tel arra, hogy milyen Ã©rtÃ©keket vehet fel az adott oszlop

DOMAIN: Ã©rtÃ©ktartomÃ¡ny egy oszlop Ã©rtÃ©keire vonatkozÃ³an

AdatmanipulÃ¡ciÃ³s lehetÅsÃ©gek Ã©s lekÃ©rdezÃ©sek

Adatok beszÃºrÃ¡sa:

Ha csak adott oszlopoknak akarunk Ã©rtÃ©ket adni (pl mert nem kÃ¶telezÅ, vagy alapÃ©rtelmezett Ã©rtÃ©k): INSERT INTO tÃ¡blanÃ©v (oszloplista) VALUES (Ã©rtÃ©klista);

Ha minden oszlop Ã©rtÃ©kÃ©t ki akarjuk tÃ¶lteni: INSERT INTO tÃ¡blanÃ©v VALUES (Ã©rtÃ©klista);

Adatok mÃ³dosÃtÃ¡sa:

UPDATE tÃ¡blanÃ©v SET oszlop=kifejezÃ©s [oszlop2=kifejezÃ©s2] [WHERE feltÃ©tel];

MÃ³dosÃtjuk egy vagy tÃ¶bb oszlop Ã©rtÃ©kÃ©t az adott tÃ¡blÃ¡ban, azokon a sorokon, amelyek eleget tesznek a WHERE zÃ¡radÃ©kban tett feltÃ©telnek.

Adatok tÃ¶rlÃ©se:

DELETE FROM tÃ¡blanÃ©v [WHERE feltÃ©tel];

TÃ¶rÃ¶ljÃ¼k az Ã¶sszes rekordot a tÃ¡blÃ¡bÃ³l, amelyek megfelelnek a WHERE zÃ¡radÃ©kban megadott feltÃ©telnek.

LekÃ©rdezÃ©sek:

SELECT oszloplista FROM tÃ¡bla;

A megadott oszlopokat kilistÃ¡zza az adott tÃ¡blÃ¡bÃ³l. oszloplista helyÃ©re megadhatÃ³ *, ha az Ã¶sszes oszlopot listÃ¡zni akarjuk.

Teljes szintaxisa:

SELECT[DISTINCT] oszloplista FROM tÃ¡blalista [WHERE feltÃ©tel] [GROUP BY oszloplista] [HAVING csoportfeltÃ©tel] [ORDER BY oszloplista [DESC]];

DISTINCT: csak a kÃ¼lÃ¶nbÃ¶zÅ sorokat Ãrja ki FROM tÃ¡blalista: a tÃ¡blalistÃ¡ban megadott tÃ¡blÃ¡kbÃ³ kÃ©pez Descartes szorzatot WHERE: kivÃ¡lasztÃ¡s a feltÃ©tel szerint GROUP BY: csoportosÃtÃ¡s az oszloplistÃ¡ban szereplÅ oszlopok szerint HAVING: a csoportosÃtÃ¡s utÃ¡n a csoportokra vonatkozÃ³ feltÃ©tel ORDER BY: az oszloplistÃ¡ban szereplÅ adatok rendezÃ©se abc szerint nÃ¶vekvÅ vagy csÃ¶kkenÅ sorrendben

Ã¶sszesÃtÅ fÃ¼ggvÃ©nyek

Leggyakrabban a GROUP BY-jal egyÃ¼tt szoktuk hasznÃ¡lni, de enÃ©lkÃ¼l is lehet. LeginkÃ¡bb a SELECT utÃ¡ni oszlolistÃ¡ban, de a where-ben Ã©s a having-ban is hasznÃ¡lhatÃ³. Az eredmÃ©nyoszlopokat AS kulcsszÃ³val el is nevezhetjÃ¼k.

MIN(oszlop): az oszlopban lÃ©vÅ minimumot adja vissza MAX(oszlop): maxot AVG(oszlop): az oszlop Ã¡tlaga SUM(oszlop): az oszlop Ã¶sszege COUNT([DISTINCT] oszlop): az eredmÃ©nyben szereplÅ (kÃ¼lÃ¶nbÃ¶zÅ) rekordok szÃ¡ma

TermÃ©szetes Ã¶sszekapcsolÃ¡s

SELECT * FROM T1, T2 WHERE T1.X = T2.X;

X az most egy oszlop, egy kulcs-kÃ¼lsÅ kulcs kapcsolat.

Erre hasznÃ¡lhatÃ³ mÃ©g SQL-ben az INNER JOIN kulcsszÃ³ is.

SELECT * FROM T1, T2 INNER JOIN T2 ON T1.X = T2.X;

HasznÃ¡lhatÃ³ mÃ©g a NATURAL JOIN kifejezÃ©s is, de ez egy picit mÃ¡shogy mÅ±kÃ¶dik. Ennek a hasznÃ¡latÃ¡hoz a kÃ©t tÃ¡bla kÃ¶zÃ¶s attribÃºtumhalmaza ugyanazokat az oszlopneveket tartalmazza mindkÃ©t tÃ¡blÃ¡ban Ã©s a pÃ¡rosÃtott oszlopok tÃpusa is megegyezik. EbbÅl kifolyÃ³lag nem kell megadnunk a kapcsolÃ³dÃ³, kulcs Ã©s kÃ¼lsÅ kulcs oszlopokat. A kÃ¶zÃ¶s oszlop csak egy pÃ©ldÃ¡nyban jelenik majd meg.

SELECT * FROM T1 NATURAL JOIN T2;

Jobboldali, baloldali Ã©s teljes kÃ¼lsÅ Ã¶sszekapcsolÃ¡s

Valamelyik, vagy mindkÃ©t tÃ¡bla Ã¶sszes rekordja szerepelni fog az eredmÃ©nyben.

Baloldali Ã¶sszekapcsolÃ¡snÃ¡l a baloldali tÃ¡bla minden rekordja megmarad, Ã©s ezekhez a rekordokhoz pÃ¡rosÃtjuk a jobboldali tÃ¡bla rekordjait. JobboldalinÃ¡l pont fordÃtva. Teljes Ã¶sszekapcsolÃ¡snÃ¡l pedig mindkÃ©t tÃ¡bla Ã¶sszes rekordja megmarad, Ã©s mindenhol a hiÃ¡nyzÃ³ helyeken NULL Ã©rtÃ©kek lesznek.

LekÃ©rdezÃ©sek eredmÃ©nyÃ©n, amikor ugyanannyi Ã©s ugyanolyan tÃpusÃº oszlopot kÃ©rÃ¼nk le, hasznÃ¡lhatunk halmazmÅ±veleteket is, pl UNION vagy INTERSECT.# 3. SimÃtÃ¡s/szÅ±rÃ©s kÃ©ptÃ©rben (Ã¡tlagolÃ³ szÅ±rÅk, Gauss simÃtÃ¡s Ã©s mediÃ¡nszÅ±rÃ©s); Ã©lek detektÃ¡lÃ¡sa (gradiens-operÃ¡torokkal Ã©s Marr-Hildreth mÃ³dszerrel)

SimÃtÃ¡s/szÅ±rÃ©s kÃ©ptÃ©rben

ÃtlagolÃ³ szÅ±rÃ©s

VesszÃ¼k egy kÃ©ppontnak egy kÃ¶rnyezetÃ©t, Ã©s vesszÃ¼k ebben a kÃ¶rnyezetben az Ã¶sszes kÃ©ppont Ã¡tlagÃ¡t. Ezzel az Ã¡tlag lesz a kÃ©ppont Ãºj Ã©rtÃ©ke. Ezt az Ã¡tlagolÃ¡st konvolÃºciÃ³val is vÃ©gezhetjÃ¼k, ahol a konvolÃºciÃ³s maszkunkban minden Ã©rtÃ©k 1/n^2, ha n*n-es a maszk.

- ez a fajta zajszÅ±rÃ©s rontja az Ã©leket
- minÃ©l nagyobb kÃ¶rnyezetet nÃ©zÃ¼nk, annÃ¡l erÅsebb a simÃtÃ³ hatÃ¡s
- haszna: csÃ¶kkenti a zajt
- kÃ¡ra: gyengÃti az Ã©leket, homÃ¡lyossÃ¡ teszi a kÃ©pet
- sÃºlyozott Ã¡tlagolÃ¡st is lehet csinÃ¡lni - konvolÃºciÃ³
  - a legnagyobb sÃºly az aktuÃ¡lis pontunknak legyen
  - ahogy tÃ¡volodunk a ponttÃ³l, annÃ¡l kisebbek legyenek a sÃºlyok

Gauss simÃtÃ¡s

- ahogy tÃ¡volodunk a ponttÃ³l, annÃ¡l kisebbek legyenek a sÃºlyok
- erre nagyon jÃ³ a gauss harang
- minden sÅ±rÅ±sÃ©gfÃ¼ggvÃ©ny integrÃ¡lja 1
  - minÃ©l nagyobb a szigma, annÃ¡l szÃ©lesebb, de annÃ¡l alacsonyabb a harang
  - ezzel szÃ©pen lehet jeleket simÃtani
- binomiÃ¡lis egyÃ¼tthatÃ³k jÃ³l kÃ¶zelÃtik a normÃ¡lis eloszlÃ¡s gÃ¶rbÃ©jÃ©t
- van 2D gauss is, harang alakÃº

hogyan lehet gauss fÃ¼ggvÃ©nyt kÃ¶zelÃteni diszkrÃ©t Ã©rtÃ©kekkel?

- vegyÃ¼k a binomiÃ¡lis egyÃ¼tthatÃ³kat tartalmazÃ³ sorvektort, Ã©s osszunk el minden elemet 2^n-nel
- ezt szorozzuk Ã¶ssze a transzponÃ¡ltjÃ¡val, Ã©s Ãgy kapjuk a gauss gÃ¶rbe kÃ¶zelÃtÃ©sÃ©t

HozzÃ¡juthatunk Ãgy diszkrÃ©t gauss eloszlÃ¡sÃº nxn-es konvolÃºciÃ³s maszkokhoz, Ã©s az ilyenekkel vett konvolÃºciÃ³ a Gauss szÅ±rÃ©s Az Ã©lek itt is rombolÃ³dnak

Lehet olyat is, hogy csak akkor simÃtunk, ha az adott kÃ©ppont intenzitÃ¡sÃ¡nak kÃ¶rnyezeti Ã¡tlagtÃ³l valÃ³ eltÃ©rÃ©se meghalad egy T kÃ¼szÃ¶bÃ©rtÃ©ket

MediÃ¡n szÅ±rÃ©s

mediÃ¡n = sorbarendezzÃ¼k az Ã©rtÃ©keket, Ã©s a kÃ¶zÃ©psÅt vesszÃ¼k min <= med <= max

mediÃ¡n nem lineÃ¡ris

mediÃ¡n szÅ±rÃ©s: nÃ©zzÃ¼k egy kÃ¶rnyezetÃ©t a pontnak, ezt rendezzÃ¼k sorba, Ã©s a kÃ¶zÃ©psÅ Ã©rtÃ©k legyen a kÃ©ppont Ãºj Ã©rtÃ©ke sÃ³-bors zaj eltÃ¼ntetÃ©sÃ©re szÃ©pen alkalmas tiszta kÃ©pet kapunk, ha pl 5x5-Ã¶s kÃ¶rnyezetben nÃ©zve a 25 kÃ©ppontbÃ³l max 12 teljesen fekete vagy teljesen fehÃ©r kÃ©ppont van megszÃ¼nteti az egyedi, Ã©s kis kiterjedÃ©sÅ± kiugrÃ¡sokat jobban megÅrzi az Ã©leket, mint az Ã¡tlagolÃ¡s nagy kiterjedÃ©sÅ± zajfoltoknÃ¡l jel-elnyomÃ³ a zajt hagyja meg, Ã©s a lÃ©nyeg tÅ±nhet el

Ãlek detektÃ¡lÃ¡sa

Ã©l ott van a kÃ©pen, ahol az intenzitÃ¡s valamilyen irÃ¡nyban felugrik, vagy lecsÃ¶kken

Ã©lek nagyon fontosak a lÃ¡tÃ¡sunban ahol markÃ¡nsak az Ã©lek, azokat jÃ³l Ã©rzÃ©keljÃ¼k

lehet ideÃ¡lis/lÃ©pcsÅs Ã©l lejtÅs Ã©l tetÅ vonal zajos

felidÃ©zÃ©s: tangens fÃ¼ggvÃ©ny tangens: Ã©rintÅ irÃ¡nytangense/meredeksÃ©ge elsÅ derivÃ¡lt: hol vannak szÃ©lsÅÃ©rtÃ©kek, monotonitÃ¡s derivÃ¡lt pozitÃv, nÅ, negatÃv, csÃ¶kken

Ã©l ott van, ahol az intenzitÃ¡sprofil elsÅ derivÃ¡ltja nagy

Gradiens operÃ¡torokkal

tÃ¶bbvÃ¡ltozÃ³s fÃ¼ggvÃ©nyeket is lehet derivÃ¡lni, pl parciÃ¡lisan egyik vÃ¡ltozÃ³t lerÃ¶gzÃtjÃ¼k, Ã©s a mÃ¡sik szerint derivÃ¡lunk gradiens - elsÅ parciÃ¡lis derivÃ¡ltakbÃ³l alkotott vektor 2D-ben az Ã©rintÅre merÅleges vektor ennek van kÃ©t komponense

gradiens nagysÃ¡ga - magnitÃºdÃ³ valamilyen vektornorma - legyen kettes norma (euklideszi norma) ekkor a gradiens kettes normÃ¡ja a komponensek nÃ©gyzetÃ¶sszegÃ©nek a nÃ©gyzetgyÃ¶ke elsÅ vektornormÃ¡nÃ¡l a gradienskomponensek abszolÃºtÃ©rtÃ©kÃ©nek az Ã¶sszegÃ©t nÃ©zzÃ¼k

```
2D-ben a kettes vektornorma az a pitagorasz tÃ©telbÅl jÃ¶n
```

2D-ben van a gradiensnek irÃ¡nya is arctan(y/x)

Ã©l irÃ¡nya a gradiensre merÅleges

diszkrÃ©t gradiens operÃ¡torok

roberts, prewitt, sobel, frei-chen

mind a nÃ©gy mÃ³dszer konvolÃºciÃ³s maszkpÃ¡rokat alkalmaz roberts operÃ¡tor adott kÃ©t 3x3-as mÃ¡trix, ha az egyikkel konvolvÃ¡lunk, akkor az x irÃ¡nyÃº parciÃ¡lis derivÃ¡ltat kÃ¶zelÃtjÃ¼k, ha a mÃ¡sikkal, akkor az y irÃ¡nyÃºt igazÃ¡bÃ³l nem is kell konvolÃºciÃ³ x: a kÃ©ppont Ã©rtÃ©kÃ©bÅl kivonjuk az Ã©szakkeleti szomszÃ©djÃ¡t y: a kÃ©ppont Ã©rtÃ©kÃ©bÅl kivonjuk az Ã©szaknyugati szomszÃ©djÃ¡t pro: kÃ¶nnyen szÃ¡mÃthatÃ³ kontra: zajÃ©rzÃ©keny

prewitt operÃ¡tor itt is kÃ©t 3x3-as maszk van, csak kicsit mÃ¡s, mint az elÅbb x: baloldali oszlop csupa 1, jobboldali csupa -1, kÃ¶zÃ©pen 0 y: felsÅ sor -1, alsÃ³ sor 1, kÃ¶zÃ©pen 0

sobel operÃ¡tor kÃ©t 3x3 maszk ha nÃ©gyzet mozaikon mintavÃ©telezett a kÃ©pÃ¼nk akkor ami kÃ©t pixel Ã©len osztozkodik (vÃ-zszintesen vagy fÃ¼ggÅlegesen szomszÃ©dos) akkor azok kÃ¶zelebb vannak egymÃ¡shoz, mintha csak csÃºcson Ã©rintkeznÃ©nek

frei-chen operÃ¡tor ugyanaz, mint a sobel, csak 2 helyett gyÃ¶k(2)

gradiens maszk tervezÃ©se x irÃ¡nyban szimmetrikus ne hÃºzzon el se balra, se jobbra asszimetrikus ne hÃºzzon el se fel, se le legyen az Ã¶sszege az elemeknek 0

8 irÃ¡nyban Ã©lt keresÅ gradiens operÃ¡torok compass operÃ¡torok

prewitt compass operÃ¡tor 8 kÃ¼lÃ¶nbÃ¶zÅ maszkkal dolgozik, a 8 Ã©gtÃ¡j irÃ¡nyÃ¡ba maszkelemek Ã¶sszege 0

robinson-3 compass operÃ¡tor 3-fÃ©le elem szerepel a maszkokban robinson-5 compass operÃ¡tor 5-fÃ©le elem

kirsch compass operÃ¡tor 0, -3, 5 Ã©rtÃ©kek szerepelnek benne

Marr-Hildreth mÃ³dszer

konvolvÃ¡ljuk a kÃ©pet egy vagy tÃ¶bb alkalmas LoG fÃ¼ggvÃ©nnyel keressÃ¼nk kÃ¶zÃ¶s nulla Ã¡tmeneteket nulla Ã¡tmenet ott van, ahol adott pont kis kÃ¶rnyezetÃ©ben elÅfordulnak pozitÃv Ã©s negatÃv Ã©rtÃ©kek is eredmÃ©nye mindig egy binÃ¡ris Ã©ltÃ©rkÃ©p lehetnek fantomÃ©lek is de ez a gyakorlatban elhanyagolhatÃ³

LoG a frekvenciatÃ©rben konvolÃºciÃ³s tÃ©tel szerint f*LoG gyorsan szÃ¡mÃthatÃ³ fourier-trafÃ³val meg pontonkÃ©nti szorzÃ¡ssal adott szigmÃ¡ra elÅre kiszÃ¡mÃthatjuk a sombrero fourier trafÃ³jÃ¡t ezt is eltÃ¡rolhatjuk# 4. AlakreprezentÃ¡ciÃ³, hatÃ¡r- Ã©s rÃ©giÃ³-alapÃº alakleÃrÃ³ jellemzÅk, Fourier leÃrÃ¡s

AlakreprezentÃ¡ciÃ³

Az alak/forma megÃtÃ©lÃ©sÃ©nek fontos szerep jut a lÃ¡tÃ¡sunkban. Az alak (shape) nem bÃr egzakt matematikai definÃciÃ³val

A szegmentÃ¡lÃ¡st kÃ¶vetÅen az objektumok kontÃºrjaibÃ³l vagy foltjaibÃ³l (attÃ³l fÃ¼ggÅen, hogy hatÃ¡r- vagy rÃ©giÃ³-alapÃº szegmentÃ¡lÃ¡st vetettÃ¼nk-e be) szÃ¡mos alakleÃrÃ³ jellemzÅt vonhatunk ki. HangsÃºlyozandÃ³, hogy itt mÃ¡r elszakadhatunk a digitÃ¡lis kÃ©pektÅl, nÃ©melyik jellemzÅ csak egy szÃ¡m, mÃ¡sok pedig Ã¶sszetett struktÃºrÃ¡k is lehetnek.

Az alakleÃrÃ³ jellemzÅket hÃ¡rom osztÃ¡lyba soroljuk.

HatÃ¡r alapÃº alakleÃrÃ³ jellemzÅk

- lÃ¡nckÃ³d, alakleÃrÃ³ szÃ¡m
- kerÃ¼let, terÃ¼let, kompaktsÃ¡g, cirkularitÃ¡s
- kÃ¶zelÃtÃ©s poligonnal
- parametrikus kontÃºr, hatÃ¡rvonal leÃrÃ³ fÃ¼ggvÃ©ny
- meredeksÃ©gi hisztogram
- gÃ¶rbÃ¼let, energia
- strukturÃ¡lis leÃrÃ¡s

Freeman fÃ©le lÃ¡nckÃ³d

- 4 vagy 8 szomszÃ©dok felÃ© mutatÃ³ vektort sorszÃ¡mozza
- Ã³ramutatÃ³ jÃ¡rÃ¡sÃ¡val ellentÃ©tes irÃ¡nyban nÃ¶vekszik

- kivÃ¡laszt a kontÃºron egy kezdÅpontot
- egymÃ¡s utÃ¡n Ãrja a kontÃºrt kÃ¶rbekÃ¶tÅ vektorok sorszÃ¡mait
- a kontÃºr leÃrhatÃ³ egy nÃ©gyes vagy nyolcas szÃ¡mrendszerbeli szÃ¡mmal, ez a lÃ¡nckÃ³d
- pro: gyors,kompakt,eltolÃ¡s-invariÃ¡ns
- kontra: nem forgÃ¡s- Ã©s skÃ¡la-invariÃ¡ns,zajÃ©rzÃ©keny,
- Mivel a lÃ¡nckÃ³d fÃ¼gg a kezdÅpont megvÃ¡lasztÃ¡sÃ¡tÃ³l, valamint nem invariÃ¡ns mÃ©g a 90 tÃ¶bbszÃ¶rÃ¶seivel valÃ³ forgatÃ¡sra sem, Ãgy bevezettÃ©k az alakleÃrÃ³ szÃ¡mot, amit a lÃ¡nckÃ³d elsÅ derivÃ¡ltjÃ¡bÃ³l kapunk.

KerÃ¼let, terÃ¼let szÃ¡mÃtÃ¡sa

- A kerÃ¼let Ã©s a terÃ¼let kÃ©t gyakran bevetett alakleÃrÃ³ jellemzÅ. MindkettÅ szÃ¡rmaztathatÃ³ a lÃ¡nckÃ³dbÃ³l is.
- 8-as lÃ¡nckÃ³d esetÃ©n: kerÃ¼let = gyÃ¶k(2) * (pÃ¡ratlan elemek szÃ¡ma) + pÃ¡ros elemek szÃ¡ma a lÃ¡nckÃ³dban
- 4-es lÃ¡nckÃ³d esetÃ©n: kerÃ¼let = lÃ¡nckÃ³d rendje (hossza)
- poligon terÃ¼lete 8-as lÃ¡nckÃ³d esetÃ©n:
    - szÃ¡montartunk egy y-t, ami kezdetben 0. Ehhez ha a lÃ¡nckÃ³dban lÃ©vÅ kÃ¶vetkezÅ szÃ¡m "felfele" mutat hozzÃ¡adunk 1-et, ha "lefele", akkor kivonunk 1-et
    - a terÃ¼letvÃ¡ltozÃ¡st szintÃ©n a lÃ¡nckÃ³dban kÃ¶vetkezÅ szÃ¡m irÃ¡nya hatÃ¡rozza meg (y alapjÃ¡n), ahogy az alÃ¡bbi kÃ©pen is lÃ¡tszik
    - a terÃ¼letet Ãºgy kapjuk, hogy foylton Ã¶sszeadogatjuk a terÃ¼letvÃ¡ltozÃ¡sokat, Ã©s a vÃ©gÃ©n vesszÃ¼k az abszulÃºtÃ©rtÃ©kÃ©t

KompaktsÃ¡g Ã©s cirkularitÃ¡s

- kompaktsÃ¡g = (kerÃ¼let)^2 / terÃ¼let
- cirkularitÃ¡s = terÃ¼let / (kerÃ¼let)^2

Parametrikus kontÃºr

- A parametrikus kontÃºr kÃ©t egyvÃ¡ltozÃ³s fÃ¼ggvÃ©nnyel reprezentÃ¡lja a szegmenst. A kontÃºron vÃ©gighaladva kÃ¶vetjÃ¼k az x Ã©s az y koordinÃ¡tÃ¡k vÃ¡ltozÃ¡sait.

RÃ©giÃ³ alapÃº alakleÃrÃ³ jellemzÅk

A hatÃ¡r-alapÃºakhoz hasonlÃ³an, szÃ¡mos rÃ©giÃ³-alapÃº alakleÃrÃ³ jellemzÅt javasoltak.

- befoglalÃ³ tÃ©glalap, rektangularitÃ¡s
- fÅtengely, mellÃ©ktengely, Ã¡tmÃ©rÅ, excentricitÃ¡s, fÅtengely szÃ¶ge
- konvex burok, konvex kiegÃ©szÃtÃ©s, konkÃ¡vitÃ¡si fa, partÃcionÃ¡lt hatÃ¡r,
- vetÃ¼letek, tÃ¶rÃ©s-kÃ¶ltsÃ©g
- topolÃ³giai leÃrÃ¡sok, Euler-szÃ¡m, szomszÃ©dsÃ¡gi fa,
- vÃ¡z,
- momentumok, invariÃ¡ns momentumok

BefoglalÃ³ tÃ©glalap, rektangularitÃ¡s

- Ã¡llÃ³ befoglalÃ³ tÃ©glalap: az objektum koordinÃ¡tÃ¡inak minimumai Ã©s maximumai megadjÃ¡k az Ã¡llÃ³ befoglalÃ³ tÃ©glalap csÃºcsait.
- minimÃ¡lis befoglalÃ³ tÃ©glalap
- rektangularitÃ¡s: Azt mondja meg, hogy az objektum âbedobozolÃ¡sakorâ mennyi a tÃ¡rgy Ã©s a âlevegÅâ Ã¡ltal elfoglalt terÃ¼letek arÃ¡nya, tehÃ¡t ---> alakzat terÃ¼lete / minimÃ¡lis befoglalÃ³ tÃ©glalap

FÅtengely, mellÃ©ktengely, Ã¡tmÃ©rÅ, excentricitÃ¡s, fÅtengely szÃ¶ge

- fÅtengely: az alakzaton belÃ¼l haladÃ³ leghosszabb egyenes szakasz
- mellÃ©ktengely: az alakzaton belÃ¼li, a fÅtengelyre merÅleges leghosszabb egyenes szakasz
- Ã¡tmÃ©rÅ: a hatÃ¡r kÃ©t legtÃ¡volabbi pontjÃ¡t kÃ¶ti Ã¶ssze. A fÅtengely hossza Ã¡ltalÃ¡ban nem egyezik meg az Ã¡tmÃ©rÅvel (csak a konvexeknÃ©l)
- excentritÃ¡s: a fÅ- Ã©s mellÃ©ktengely hosszarÃ¡nya---> d1/d2
- fÅtengely szÃ¶ge: a fÅtengely Ã©s az x-tengely Ã¡ltal bezÃ¡rt szÃ¶g

Konvex burok, konvex kiegÃ©szÃtÃ©s, konkÃ¡vitÃ¡si fa, partÃcionÃ¡lt hatÃ¡r

- konvex burok: az alakzatot tartalmazÃ³ minimÃ¡lis konvex alakzat

- konvex kiegÃ©szÃtÃ©s: a konvex burok Ã©s az alakzat kÃ¼lÃ¶nbsÃ©ge
- konkÃ¡vitÃ¡si fa: A fa gyÃ¶kere a kiindulÃ¡si alakzat, az elsÅ szinten a konvex kÃ¼lÃ¶nbsÃ©g alakzatai helyezkednek el, melyekre a faÃ©pÃtÃ©st rekurzÃv mÃ³don folytatjuk.
- partÃcionÃ¡lt hatÃ¡r: A konvex burok hatÃ¡rÃ¡t osztja fel rÃ©szekre.

VetÃ¼letek, tÃ¶rÃ©s-kÃ¶ltsÃ©g

- vetÃ¼letek: A binÃ¡ris kÃ©pekbÅl kÃ©pzett nem-negatÃv egÃ©szekbÅl Ã¡llÃ³ (1D) tÃ¶mbÃ¶k.
- tÃ¶rÃ©s-kÃ¶ltsÃ©g: A vetÃ¼letek tovÃ¡bbragozÃ¡sa, kiszÅ±ri a zajos kÃ©pek oszlopaiban lÃ©vÅ âmagÃ¡nyosâ objektumpontokat.

TopolÃ³giai leÃrÃ¡sok, Euler-szÃ¡m, szomszÃ©dsÃ¡gi fa,

- topolÃ³giai leÃrÃ¡sok
    - binÃ¡ris kÃ©p: kÃ©tfÃ©le Ã©rtÃ©k lehet benne, az 1-es az alakzatot (komponenst) reprezentÃ¡lja feketÃ©vel, mÃg a 0-s a hÃ¡tteret(lyukakat) fehÃ©rrel
    - komponens: maximÃ¡lisan Ã¶sszefÃ¼ggÅ fekete halmaz
    - Ã¼reg: a negÃ¡lt kÃ©p egy vÃ©ges komponense
- Euler-fÃ©le szÃ¡m: egyetlen egÃ©sz szÃ¡m ---> komponensek szÃ¡ma - Ã¼regek szÃ¡ma, rengeteg kÃ©pre lehet az ugyanaz. Valamit elÃ¡rul a kÃ©prÅl, de Ã¶nmagÃ¡ban keveset.
- Ã¶sszefÃ¼ggÅsÃ©gi-fa: A binÃ¡ris kÃ©pekhez rendelt irÃ¡nyÃtott grÃ¡f
    - minden egyes csÃºcs megfelel a kÃ©p egy (fehÃ©r vagy fekete) komponensÃ©nek,
    - a grÃ¡f tartalmazza az (X,Y) Ã©let, ha az X komponens âkÃ¶rÃ¼lveszià a vele szomszÃ©dos Y komponenst

VÃ¡z

A vÃ¡z egy gyakran alkalmazott rÃ©giÃ³-alapÃº alakleÃrÃ³ jellemzÅ, mely leÃrja az objektumok Ã¡ltalÃ¡nos formÃ¡jÃ¡t.\ AlapvetÅen 3-fÃ©lekÃ©pp hatÃ¡rozhatjuk meg: - a vÃ¡zat az objektum azon pontjai alkotjÃ¡k, melyekre kettÅ vagy tÃ¶bb legkÃ¶zelebbi hatÃ¡rpont talÃ¡lhatÃ³. - Az objektum hatÃ¡rÃ¡t (minden pontjÃ¡ban) egyidejÅ±leg felgyÃºjtjuk. A vÃ¡z azokbÃ³l a pontokbÃ³l Ã¡ll, ahol a tÅ±zfrontok talÃ¡lkoznak Ã©s kioltjÃ¡k egymÃ¡st. (FeltÃ©telezzÃ¼k, hogy a tÅ±zfrontok minden irÃ¡nyban egyenletes sebessÃ©ggel, vagyis izotropikusan terjednek.) - A vÃ¡zat az objektumba beÃrhatÃ³ maximÃ¡lis (nyÃlt) hipergÃ¶mbÃ¶k kÃ¶zÃ©ppontjai alkotjÃ¡k. Egy beÃrhatÃ³ hipergÃ¶mb maximÃ¡lis, ha Åt nem tartalmazza egyetlen mÃ¡sik beÃrhatÃ³ hipergÃ¶mb sem.

InvariÃ¡ns az eltolÃ¡sra, elforgatÃ¡sra Ã©s az uniform skÃ¡lÃ¡zÃ¡sra

Momentumok, invariÃ¡ns momentumok

Pro: szÃ¡mok, tÃ¶bbszintÅ± kÃ©pekre is Ã©rtelmezettek, invariÃ¡nsak a fÅbb geometriai mÅ±veletekre\ Bizonyos (centrÃ¡lis) momentumoknak geometriai jelentÃ©s is tulajdonÃthatÃ³, illetve fontos jellemzÅk kifejezhetÅk a segÃtsÃ©gÃ¼kkel, pÃ©ldÃ¡ul sÃºlypont.

Javasoltak viszont 7 Ãºn. invariÃ¡ns momentumot is (ld. 56. dia), amelyekhez nem tÃ¡rsÃthatÃ³k kÃ¼lÃ¶nÃ¶sebb jelentÃ©sek, de a belÅlÃ¼k alkotott rendezett hetesek (vagy akÃ¡r hÃ¡rmasok, ha nem vesszÃ¼k mindet figyelembe) jÃ³l jellemzik az objektumokat.

Fourier leÃrÃ¡s

Ez egy transzformÃ¡ciÃ³n alapulÃ³ alakleÃrÃ¡s

TranszformÃ¡ljuk (hangsÃºlyozandÃ³, hogy habÃ¡r 2D kÃ©pek szegmenseit jellemezzÃ¼k, itt szigorÃºan 1D Fourier transzformÃ¡ciÃ³t alkalmazunk) a hatÃ¡r K darab mintavÃ©telezett pontjÃ¡bÃ³l (mint komplex s(k) szÃ¡mokbÃ³l) kÃ©pzett **s** vektort. Az eredmÃ©nyÃ¼l kapott **a** vektor (komplex a(k) egyÃ¼tthatÃ³k) adjÃ¡k a Fourier leÃrÃ¡st (vagyis tartalmazza a Fourier egyÃ¼tthatÃ³kat, a transzformÃ¡ciÃ³ bÃ¡zisfÃ¼ggvÃ©nyeinek sÃºlyait). Az alakzat rekonstrukciÃ³jÃ¡hoz az inverz Fourier-transzformÃ¡ciÃ³t kell vÃ©grehajtani.

A K darab Fourier egyÃ¼tthatÃ³bÃ³l visszakaphatnÃ¡nk torzÃtatlanul az eredeti mitnavÃ©telezett pontokat, az alakleÃrÃ¡shoz viszont nem az Ã¶sszes sÃºlyt, hanem csak egy rÃ©szÃ¼ket tartjuk meg, mindÃ¶ssze P<K darab egyÃ¼tthatÃ³ alapjÃ¡n tÃ©rÃ¼nk vissza a kÃ©ptÃ©rbe - ekkor a kÃ©ptÃ©rben ismÃ©t K darab pontot kapunk vissza, de nem a kiindulÃ¡s mintavÃ©telezettjeit. - Az egyÃ¼tthatÃ³k egy rÃ©szÃ©nek eldobÃ¡sÃ¡val kapott leÃrÃ¡s (a meghagyott egyÃ¼tthatÃ³k adjÃ¡k a jellemzÃ©st) voltakÃ©ppen egy vesztesÃ©ges tÃ¶mÃ¶rÃtÃ©s: kevesebb adattal tudjuk jÃ³l-rosszul kÃ¶zelÃteni a kiindulÃ¡sit.

Az alÃ¡bbi kÃ©p azt mutatja, hogy hogy a 64 kontÃºrponttal mintavÃ©telezett nÃ©gyzetre csak sok egyÃ¼tthatÃ³ megtartÃ¡sÃ¡val tudunk nÃ©gyzetfÃ©lÃ©t rekonstruÃ¡lni.

A kÃ¶vetkezÅ kÃ©pen viszont tesztobjektum hatÃ¡ra kÃ¶zel 3000 ponttal adott, Ã©s mÃ¡r 36 egyÃ¼tthatÃ³ is visszaadhat 3000 pontot Ãºgy, hogy azok jÃ³l kÃ¶zelÃtik a kiindulÃ¡si kontÃºrt. # 5. Algoritmusok vezÃ©rlÃ©si szerkezetei Ã©s megvalÃ³sÃtÃ¡suk C programozÃ¡si nyelven. A szekvenciÃ¡lis, iterÃ¡ciÃ³s, elÃ¡gazÃ¡sos, Ã©s az eljÃ¡rÃ¡s vezÃ©rlÃ©s

Algoritmusok vezÃ©rlÃ©si szerkezetei Ã©s megvalÃ³sÃtÃ¡suk C nyelven

Algoritmus: bÃ¡rmilyen jÃ³l definiÃ¡lt szÃ¡mÃtÃ¡si eljÃ¡rÃ¡st, amely bemenetkÃ©nt bizonyos Ã©rtÃ©ket vagy Ã©rtÃ©keket kap Ã©s kimenetkÃ©nt bizonyos Ã©rtÃ©ket vagy Ã©rtÃ©keket Ã¡llÃt elÅ. VizsgÃ¡lhatjuk helyessÃ©g, idÅ- Ã©s tÃ¡rigÃ©ny szempontjÃ¡bÃ³l

Algoritmus vezÃ©rlÃ©se: Az az eloÌiÌraÌs, amely az algoritmus minden leÌpeÌseÌre (reÌszmuÌveleteÌre) kijeloÌli, hogy a leÌpeÌs veÌgrehajtaÌsa utaÌn melyik leÌpeÌs veÌgrehajtaÌsaÌval folytatoÌdjon (esetleg fejezoÌdjeÌk be) az algoritmus veÌgrehajtaÌsa. Az algoritmusnak, mint mÅ±veletnek a vezÃ©rlÃ©s a legfontosabb komponense.

NÃ©gy fÅ vezÃ©rlÃ©si mÃ³dot kÃ¼lÃ¶nbÃ¶ztetÃ¼nk meg: - SzekvenciÃ¡lis: VÃ©ges sok adott mÅ±velet rÃ¶gzÃtett sorrendben egymÃ¡s utÃ¡n tÃ¶rtÃ©nÅ vÃ©grehajtÃ¡sa. (sorban egymÃ¡s utÃ¡n) - SzelekciÃ³s: VÃ©ges sok rÃ¶gzÃtett mÅ±velet kÃ¶zÃ¼l adott feltÃ©tel alapjÃ¡n valamelyik vÃ©grehajtÃ¡sa. (if, else, if else, switch) - IsmÃ©tlÃ©ses: Adott mÅ±velet adott feltÃ©tel szerinti ismÃ©telt vÃ©grehajtÃ¡sa. (for, while, do while) - EljÃ¡rÃ¡s: Adott mÅ±velet alkalmazÃ¡sa adott argumentumokra, ami az argumentumok Ã©rtÃ©kÃ©nek pontosan meghatÃ¡rozott vÃ¡ltozÃ¡sÃ¡t eredmÃ©nyezi. (void func, int func, double func, â¦)

A vezÃ©rlÃ©si mÃ³dok nyelvek feletti fogalmak.

A imperatÃv (algoritmikus) programozÃ¡si nyelvekben ezek a vezÃ©rlÃ©si szerkezetek (kÃ¶zvetlenÃ¼l vagy kÃ¶zvetve) megvalÃ³sÃthatÃ³k.

A szekvenciÃ¡lis, iterÃ¡ciÃ³s, elÃ¡gazÃ¡sos, Ã©s az eljÃ¡rÃ¡s vezÃ©rlÃ©s

SzekvenciÃ¡lis vezÃ©rlÃ©s

SzekvenciÃ¡lis vezÃ©rlÃ©srÅl akkor beszÃ©lÃ¼nk, amikor a P problÃ©ma megoldÃ¡sÃ¡t Ãºgy kapjuk, hogy a problÃ©mÃ¡t P1,..., Pn rÃ©szproblÃ©mÃ¡kra bontjuk, majd az ezekre adott megoldÃ¡sokat (rÃ©szalgoritmusokat) sorban egymÃ¡s utÃ¡n hajtjuk vÃ©gre.

P1,..., Pn lehetnek elemi mÅ±veletek, vagy nem elemi rÃ©szproblÃ©mÃ¡k megnevezÃ©sei.

EljÃ¡rÃ¡svezÃ©rlÃ©s

EljÃ¡rÃ¡svezÃ©rlÃ©srÅl akkor beszÃ©lÃ¼nk, amikor egy mÅ±veletet adott argumentumokra alkalmazunk, aminek hatÃ¡sÃ¡ra az argumentumok Ã©rtÃ©kei pontosan meghatÃ¡rozott mÃ³don vÃ¡ltoznak meg.

Az eljÃ¡rÃ¡svezÃ©rlÃ©s fajtÃ¡i:

- EljÃ¡rÃ¡smÅ±velet
- FÃ¼ggvÃ©nymÅ±velet

C-ben kicsi a kÃ¼lÃ¶nbsÃ©g a kettÅ kÃ¶zÃ¶tt.

FÃ¼ggvÃ©nymÅ±velet

- A matematikai fÃ¼ggvÃ©ny fogalmÃ¡nak Ã¡ltalÃ¡nosÃtÃ¡sa
- Ha egy rÃ©szproblÃ©ma cÃ©lja egy Ã©rtÃ©k kiszÃ¡mÃtÃ¡sa adott Ã©rtÃ©kek fÃ¼ggvÃ©nyÃ©ben, akkor a megoldÃ¡st megfogalmazhatjuk fÃ¼ggvÃ©nymÅ±velettel.
- A fÃ¼ggvÃ©nymÅ±velet specifikÃ¡ciÃ³ja tartalmazza:
    - A mÅ±velet elnevezÃ©sÃ©t
    - A paramÃ©terek felsorolÃ¡sÃ¡t
    - Mindegyik paramÃ©ter adattÃpusÃ¡t
    - A mÅ±velet hatÃ¡sÃ¡nak leÃrÃ¡sÃ¡t
    - A fÃ¼ggvÃ©nymÅ±velet eredmÃ©nytÃpusÃ¡t
- Minden fÃ¼ggvÃ©nyben szerepelnie kell legalÃ¡bb egy return utasÃtÃ¡snak
- Ha a fÃ¼ggvÃ©nyben egy ilyen utasÃtÃ¡st hajtunk vÃ©gre, akkor a fÃ¼ggvÃ©ny Ã©rtÃ©kÃ©nek kiszÃ¡mÃtÃ¡sa befejezÅdik. A hÃvÃ¡s helyÃ©n a fÃ¼ggvÃ©ny a return Ã¡ltal kiszÃ¡mÃtott Ã©rtÃ©ket veszi fel

EljÃ¡rÃ¡smÅ±velet

- Ha eljÃ¡rÃ¡st szeretnÃ©nk kÃ©szÃteni C nyelven, akkor egy olyan fÃ¼ggvÃ©nyt kell deklarÃ¡lni, melynek eredmÃ©nytÃpusa void. Ebben az esetben a fÃ¼ggvÃ©ny definÃciÃ³jÃ¡ban nem kÃ¶telezÅ a return utasÃtÃ¡s, illetve ha mÃ©gis van ilyen, akkor nem adhatÃ³ meg utÃ¡na kifejezÃ©s

MegvalÃ³sÃtÃ¡s

- csak bemenÅ mÃ³dÃº argumentumok vannak
- pointerekkel lehet kezelni kimenÅ argumentumokkÃ©nt is

SzelekciÃ³s vezÃ©rlÃ©s

SzelekcioÌs vezeÌrleÌsroÌl akkor beszeÌluÌnk, amikor veÌges sok roÌgziÌtett muÌvelet koÌzuÌl veÌges sok felteÌtel alapjaÌn vaÌlasztjuk ki, hogy melyik muÌvelet keruÌljoÌn veÌgrehajtaÌsra.

TiÌpusai:

- EgyszeruÌ szelekcioÌs vezeÌrleÌs
- ToÌbbszoÌroÌs szelekcioÌs vezeÌrleÌs
- EsetkivaÌlasztaÌsos szelekcioÌs vezeÌrleÌs
- A fenti haÌrom âegyeÌbkeÌntâ aÌggal

EgyszerÅ± szelekciÃ³s vezÃ©rlÃ©s

- EgyszeruÌ szelekcioÌ eseteÌn egyetlen felteÌtel eÌs egyetlen muÌvelet van (ami persze lehet oÌsszetett).
- A vezÃ©rlÃ©s bÅvÃthetÅ Ãºgy, hogy a 3. pontban Ã¼res mÅ±velet helyett egy B mÅ±veletet hajtunk vÃ©gre, ekkor beszÃ©lÃ¼nk egyÃ©bkÃ©nt Ã¡grÃ³l.

EgyszerÅ± szelekciÃ³s utasÃtÃ¡s megvalÃ³sÃtÃ¡sa C nyelven:

```
if(F) { A; }
```

TÃ¶bbszÃ¶rÃ¶s szelekciÃ³s vezÃ©rlÃ©s

- Ha toÌbb felteÌteluÌnk eÌs toÌbb muÌveletuÌnk van, akkor toÌbbszoÌroÌs szelekcioÌroÌl beszeÌluÌnk.
- A toÌbbszoÌroÌs szelekcioÌ is boÌviÌthetoÌ egyeÌbkeÌnt aÌggal uÌgy, hogy egy nemuÌres B muÌveletet hajtunk veÌgre a 3. leÌpeÌsben.
- Legyenek Fi logikai kifejezeÌsek, Ai (Ã©s B) pedig tetszoÌleges muÌveletek. Az Fi felteÌtelekboÌl eÌs Ai (Ã©s B) muÌveletekboÌl keÌpzett toÌbbszoÌroÌs szelekcioÌs vezeÌrleÌs a koÌvetkezoÌ vezeÌrleÌsi eloÌiÌraÌst jelenti:
  - Az Fi felteÌtelek sorban toÌrteÌnoÌ kieÌrteÌkeleÌseÌvel adjunk vaÌlaszt a koÌvetkezoÌ keÌrdeÌsre: Van-e olyan i amelyre teljesuÌl, hogy az Fi felteÌtel igaz eÌs az oÌsszes Fj felteÌtel hamis?
  - Ha van ilyen i, akkor hajtsuk veÌgre az Ai muÌveletet eÌs fejezzuÌk be az oÌsszetett muÌvelet veÌgrehajtaÌsaÌt.
  - EgyeÌbkeÌnt, vagyis ha minden Fi felteÌtel hamis, akkor (hajtsuk vÃ©gre B-t Ã©s) fejezzuÌk be az oÌsszetett muÌvelet veÌgrehajtaÌsaÌt.

TÃ¶bbszÃ¶rÃ¶s szelekciÃ³s utasÃtÃ¡s megvalÃ³sÃtÃ¡sa C nyelven:

```
if(F1) { A1; } else if (F2) { A2; }...
```

- C nyelvben nincs kuÌloÌn utasiÌtaÌs a toÌbbszoÌroÌs szelekcioÌ megvaloÌsiÌtaÌsaÌra, ezeÌrt az egyszeruÌ szelekcioÌ ismeÌtelt alkalmazaÌsaÌval kell azt megvaloÌsiÌtani.
- Ez azon az oÌsszefuÌggeÌsen alapszik, hogy a toÌbbszoÌroÌs szelekcioÌ levezethetoÌ egyszeruÌ szelekcioÌk megfeleloÌ oÌsszeteÌteleÌvel.

EsetkivÃ¡lasztÃ³s szelekciÃ³s vezÃ©rlÃ©s

Ha a toÌbbszoÌroÌs szelekcioÌs vezeÌrleÌsben minden Fi felteÌteluÌnk K â Hi alakuÌ, akkor esetkivaÌlasztaÌsos szelekcioÌroÌl beszeÌluÌnk.

- Legyen K egy adott tiÌpusuÌ kifejezeÌs, legyenek Hi ilyen tiÌpusuÌ halmazok, Ai (Ã©s B) pedig tetszoÌleges muÌveletek. A K szelektor kifejezeÌsboÌl, Hi kivaÌlasztoÌ halmazokboÌl eÌs Ai (Ã©s B) muÌveletekboÌl keÌpzett esetkivaÌlasztaÌsos szelekcioÌs vezeÌrleÌs a koÌvetkezoÌ vezeÌrleÌsi eloÌiÌraÌst jelenti:

  - EÌrteÌkeljuÌk ki a K kifejezeÌst eÌs folytassuk a 2.) leÌpeÌssel.
  - Adjunk vaÌlaszt a koÌvetkezoÌ keÌrdeÌsre: Van-e olyan i (1<=i<=n), amelyre teljesÃ¼l, hogy a kiszÃ¡molt Ã©rtÃ©k eleme a Hi halmaznak Ã©s nem eleme az Ã¶sszes Hj (1<=j<i) halmaznak?
  - Ha van ilyen i, akkor hajtsuk veÌgre az Ai muÌveletet eÌs fejezzuÌk be az oÌsszetett muÌvelet veÌgrehajtaÌsaÌt.
  - EgyeÌbkeÌnt, vagyis ha K nem eleme egyetlen Hi halmaznak sem, akkor (hajtsuk vÃ©gre B-t Ã©s) fejezzuÌk be az oÌsszetett muÌvelet veÌgrehajtaÌsaÌt.

- A kivaÌlasztoÌ halmazok megadaÌsa az esetkivaÌlasztaÌsos szelekcioÌ kritikus pontja.

- Algoritmusok tervezeÌse soraÌn baÌrmilyen effektiÌv halmazmegadaÌst hasznaÌlhatunk, azonban a teÌnyleges megvaloÌsiÌtaÌskor csak a vaÌlasztott programozaÌsi nyelv eszkoÌzeit alkalmazhatjuk.

A switch utasÃtÃ¡s: Ha egy kifejezeÌs eÌrteÌke alapjaÌn toÌbbfeÌle utasiÌtaÌs koÌzuÌl kell vaÌlasztanunk, a switch utasiÌtaÌst hasznaÌlhatjuk. Megadhatjuk, hogy hol kezdoÌdjoÌn eÌs meddig tartson az utasiÌtaÌs-sorozat veÌgrehajtaÌsa. A switch utasiÌtaÌs szintaxisa C-ben:

```
switch(kifejezÃ©s) { case konstans1: A; break; case konstans2: B; break; default: D; }
```

- A szelektor kifejezeÌs eÌs a konstansok tiÌpusaÌnak meg kell egyeznie. Egy konstans legfeljebb egy case moÌgoÌtt eÌs a default kulcsszoÌ is legfeljebb egyszer szerepelhet egy switch utasiÌtaÌsban.
- A default cimke olyan, mintha a szelektor kifejezeÌs lehetseÌges eÌrteÌkei koÌzuÌl minden olyat felsorolnaÌnk, ami nem szerepel case moÌgoÌtt az adott switch-ben.
- A cimkeÌk (beleeÌrtve a default-ot is) sorrendje tetszoÌleges lehet, az nem befolyaÌsolja, hogy a szelektor kifejezeÌs melyik cimkeÌt vaÌlasztja.

- A szelektor kifejezeÌs eÌrteÌkeÌtoÌl csak az fuÌgg, hogy melyik helyen kezdjuÌk el veÌgrehajtani a switch magjaÌt. Ha a veÌgrehajtaÌs elkezdoÌdik, akkor onnantoÌl kezdve az elsoÌ break (vagy return) utasiÌtaÌsig, vagy a switch veÌgeÌig sorban hajtoÌdnak veÌgre az utasiÌtaÌsok. Ebben a faÌzisban a tovaÌbbi case eÌs default cimkeÌknek maÌr nincs jelentoÌsseÌge.
- A Hi halmazok elemszaÌma tetszoÌleges lehet, viszont a case-ek utaÌn csak egy-egy eÌrteÌk aÌllhat.

IsmÃ©tlÃ©ses vezÃ©rlÃ©sek

IsmeÌtleÌses vezeÌrleÌsen olyan vezeÌrleÌsi eloÌiÌraÌst eÌrtuÌnk, amely adott muÌveletnek adott felteÌtel szerinti ismeÌtelt veÌgrehajtaÌsaÌt iÌrja eloÌ.

Az algoritmustervezeÌs soraÌn a leginkaÌbb megfeleloÌ ismeÌtleÌses vezeÌrleÌsi formaÌt hasznaÌljuk, fuÌggetlenuÌl attoÌl, hogy a megvaloÌsiÌtaÌsra hasznaÌlt programozaÌsi nyelvben koÌzvetlenuÌl megvaloÌsiÌthatoÌ-e ez a vezeÌrleÌsi moÌd.

IsmeÌtleÌses vezeÌrleÌs keÌpzeÌseÌt ciklusszervezeÌsnek is nevezik, iÌgy az ismeÌtleÌsben szereploÌ muÌveletet ciklusmagnak hiÌvjuk.

Az ismeÌtleÌsi felteÌtel szerint oÌtfeÌle ismeÌtleÌses vezeÌrleÌst kuÌloÌnboÌztetuÌnk meg:

- KezdoÌfelteÌteles
- VeÌgfelteÌteles
- SzaÌmlaÌlaÌsos
- Hurok
- DiszkreÌt

KezdÅfeltÃ©teles ismÃ©tlÃ©ses vezÃ©rlÃ©s

KezdoÌfelteÌteles vezeÌrleÌsroÌl akkor beszeÌluÌnk, ha a ciklusmag (ismeÌtelt) veÌgrehajtaÌsaÌt egy beleÌpeÌsi (ismeÌtleÌsi) felteÌtelhez koÌtjuÌk.

- Legyen F logikai kifejezeÌs, M pedig tetszoÌleges muÌvelet. Az F ismeÌtleÌsi felteÌtelboÌl eÌs az M muÌveletboÌl (a ciklusmagboÌl) keÌpzett kezdoÌfelteÌteles ismeÌtleÌses vezeÌrleÌs a koÌvetkezoÌ vezeÌrleÌsi eloÌiÌraÌst jelenti:

    - EÌrteÌkeljuÌk ki az F felteÌtelt eÌs folytassuk a 2.) leÌpeÌssel.
    - Ha F eÌrteÌke hamis, akkor az ismeÌtleÌs eÌs ezzel egyuÌtt az oÌsszetett muÌvelet veÌgrehajtaÌsa befejezoÌdoÌtt.
    - EgyeÌbkeÌnt, vagyis ha az F eÌrteÌke igaz, akkor hajtsuk veÌgre az M muÌveletet, majd folytassuk az 1.) leÌpeÌssel.

- A felteÌtel ellenoÌrzeÌse a muÌvelet eloÌtt toÌrteÌnik

- Ha az F Ã©rtÃ©ke kezdetben hamis, az Ã¶sszetett mÅ±velet vÃ©grehajtÃ¡sa befejezÅdik anÃ©lkÃ¼l, hogy az M mÅ±velet egyszer is vÃ©grehajtÃ¡sra kerÃ¼lne
- Ha az F Ã©rtÃ©ke igaz, Ã©s az M mÅ±velet nincs hatÃ¡ssal az F feltÃ©telre, akkor F igaz is marad, tehÃ¡t az Ã¶sszetett mÅ±velet vÃ©grehajtÃ¡sa nem tud befejezÅdni. Ilyenkor vÃ©gtelen ciklus vÃ©grehajtÃ¡sÃ¡t Ãrtuk elÅ.
- Fontos tehÃ¡t, hogy az M mÅ±velet hatÃ¡ssal legyen az F feltÃ©telre.

A while utasÃtÃ¡s: Ha valamilyen muÌveletet mindaddig veÌgre kell hajtani, amiÌg egy felteÌtel igaz, a while utasiÌtaÌs hasznaÌlhatoÌ.

```
while(F) { M; }
```

VÃ©gfeltÃ©teles ismÃ©tlÃ©ses vezÃ©rlÃ©s

A vÃ©gfeltÃ©teles ismÃ©tlÃ©ses vezÃ©rlÃ©s alapvetÅen abban kÃ¼lÃ¶nbÃ¶zik a kezdÅfeltÃ©teles ismÃ©tlÃ©ses vezÃ©rlÃ©stÅl, hogy a ciklusmag legalÃ¡bb egyszer vÃ©grehajtÃ³dik. VeÌgfelteÌteles vezeÌrleÌsroÌl akkor beszeÌluÌnk, ha a ciklusmag elhagyaÌsaÌt egy kileÌpeÌsi felteÌtelhez koÌtjuÌk.

- Legyen F logikai kifejezeÌs, M pedig tetszoÌleges muÌvelet. Az F kileÌpeÌsi felteÌtelboÌl eÌs az M muÌveletboÌl (a ciklusmagboÌl) keÌpzett veÌgfelteÌteles ismeÌtleÌses vezeÌrleÌs a koÌvetkezoÌ vezeÌrleÌsi eloÌiÌraÌst jelenti:
    - Hajtsuk veÌgre az M muÌveletet majd folytassuk a 2.) leÌpeÌssel.
    - EÌrteÌkeljuÌk ki az F felteÌtelt eÌs folytassuk a 3.) leÌpeÌssel.
    - Ha F eÌrteÌke igaz, akkor az ismeÌtleÌses vezeÌrleÌs eÌs ezzel egyuÌtt az oÌsszetett muÌvelet veÌgrehajtaÌsa befejezoÌdoÌtt.
    - EgyeÌbkeÌnt, vagyis ha az F eÌrteÌke hamis, akkor folytassuk az 1.) leÌpeÌssel.
- Ha az F eÌrteÌke kezdetben hamis, eÌs az M muÌvelet nincs hataÌssal F-re, akkor veÌgtelen ciklust kapunk. Ha az F eÌrteÌke kezdetben igaz, M legalaÌbb egyszer akkor is veÌgrehajtaÌsra keruÌl.
- A kezdoÌ eÌs veÌgfelteÌteles vezeÌrleÌsek kifejezhetoÌek egymaÌs segiÌtseÌgeÌvel.

A do while: utasÃtÃ¡s Ha valamilyen muÌveletet mindaddig veÌgre kell hajtani, amiÌg egy felteÌtel igaz, a do while utasiÌtaÌs hasznaÌlhatoÌ. A muÌvelet veÌgrehajtaÌsa szuÌkseÌges a felteÌtel kieÌrteÌkeleÌseÌhez. A felteÌtel ellenoÌrzeÌse a muÌvelet utaÌn toÌrteÌnik, iÌgy ha a felteÌtel kezdetben hamis volt, a muÌveletet akkor is legalaÌbb egyszer veÌgrehajtjuk.

```
do { M; } while (!F);
```

SzÃ¡mlÃ¡lÃ¡sos ismÃ©tlÃ©ses vezÃ©rlÃ©sek

SzaÌmlaÌlaÌsos ismeÌtleÌses vezeÌrleÌsroÌl akkor beszeÌluÌnk, ha a ciklusmagot veÌgre kell hajtani sorban minden olyan eÌrteÌkeÌre (noÌvekvoÌ vagy csoÌkkenoÌ sorrendben), amely egy adott intervallumba esik.

Legyen a eÌs b egeÌsz eÌrteÌk, i egeÌsz tiÌpusuÌ vaÌltozoÌ, M pedig tetszoÌleges muÌvelet, amelynek nincs hataÌsa a, b eÌs i eÌrteÌkeÌre.

NÃ¶vekvÅ szÃ¡mlÃ¡lÃ¡sos ismÃ©tlÃ©ses vezÃ©rlÃ©sek:

- Az a eÌs b hataÌreÌrteÌkekboÌl, i ciklusvaÌltozoÌboÌl eÌs M muÌveletboÌl (ciklusmagboÌl) keÌpzett noÌvekvoÌ szaÌmlaÌlaÌsos ismeÌtleÌses vezeÌrleÌs az alaÌbbi vezeÌrleÌsi eloÌiÌraÌst jelenti:
    - Legyen i = a eÌs folytassuk a 2.) leÌpeÌssel.
    - Ha b < i (i nagyobb mint a intervallum vÃ©gpontja), akkor az ismeÌtleÌs eÌs ezzel egyuÌtt az oÌsszetett muÌvelet veÌgrehajtaÌsa befejezoÌdoÌtt.
    - EgyeÌbkeÌnt, vagyis ha i â¤ b, akkor hajtsuk veÌgre az M muÌveletet, majd folytassuk a 4.) leÌpeÌssel.
    - NoÌveljuÌk i eÌrteÌkeÌt 1-gyel, eÌs folytassuk a 2.) leÌpeÌssel.

CsÃ¶kkenÅ szÃ¡mlÃ¡lÃ¡sos ismÃ©tlÃ©ses vezÃ©rlÃ©sek:

- Az a eÌs b hataÌreÌrteÌkekboÌl, i ciklusvaÌltozoÌboÌl eÌs M muÌveletboÌl (ciklusmagboÌl) keÌpzett csoÌkkenoÌ szaÌmlaÌlaÌsos ismeÌtleÌses vezeÌrleÌs az alaÌbbi vezeÌrleÌsi eloÌiÌraÌst jelenti:
    - Legyen i = b eÌs folytassuk a 2.) leÌpeÌssel.
    - Ha i < a, akkor az ismeÌtleÌs eÌs ezzel egyuÌtt az oÌsszetett muÌvelet veÌgrehajtaÌsa befejezoÌdoÌtt.
    - EgyeÌbkeÌnt, vagyis ha a â¤ i, akkor hajtsuk veÌgre az M muÌveletet, majd folytassuk a 4.) leÌpeÌssel.
    - CsoÌkkentsuÌk i eÌrteÌkeÌt 1-gyel, eÌs folytassuk a 2.) leÌpeÌssel.

A for utasÃtÃ¡s: Ha valamilyen muÌveletet sorban toÌbb eÌrteÌkeÌre is veÌgre kell hajtani, akkor a for utasiÌtaÌs hasznaÌlhatoÌ.

```
for (i = a; i <=b; i++) { M; } for (kif1; kif2; kif3) { M; }
```

C-ben a for utasÃtÃ¡s Ã¡ltalÃ¡nos alakja: - A kif1 eÌs kif3 toÌbbnyire eÌrteÌkadaÌs vagy fuÌggveÌnyhiÌvaÌs, kif2 pedig relaÌcioÌs kifejezeÌs. - BaÌrmelyik kifejezeÌs elhagyhatoÌ, de a pontosvesszoÌknek meg kell maradniuk - kif2 elhagyaÌsa eseteÌn a felteÌtelt konstans igaznak tekintjuÌk, ekkor a break vagy return segiÌtseÌgeÌvel lehet kiugrani a ciklusboÌl.

Hurok ismÃ©tlÃ©ses vezÃ©rlÃ©s

Amikor a ciklusmag ismeÌtleÌseÌt a ciklusmagon beluÌl vezeÌreljuÌk uÌgy, hogy a ciklus kuÌloÌnboÌzoÌ pontjain adott felteÌtelek teljesuÌleÌse eseteÌn a ciklus veÌgrehajtaÌsaÌt befejezzuÌk, hurok ismeÌtleÌses vezeÌrleÌsroÌl beszeÌluÌnk.

- Legyenek Fi logikai kifejezeÌsek, Ki eÌs Mj pedig tetszoÌleges (akaÌr uÌres) muÌveletek 1â¤iâ¤n eÌs 0â¤jâ¤n Ã©rteÌkekre. Az Fi kijaÌrati felteÌtelekboÌl, Ki kijaÌrati muÌveletekboÌl eÌs az Mi muÌveletekboÌl keÌpzett hurok ismeÌtleÌses vezeÌrleÌs a koÌvetkezoÌ eloÌiÌraÌst jelenti:
    - Az ismeÌtleÌses vezeÌrleÌs koÌvetkezoÌ veÌgrehajtandoÌ egyseÌge az M0 muÌvelet.
    - Ha a veÌgrehajtandoÌ egyseÌg az Mj muÌvelet, akkor ez veÌgrehajtoÌdik. j = n eseteÌn folytassuk az 1.) leÌpeÌssel, kuÌloÌnben pedig az Fj+1 felteÌtel veÌgrehajtaÌsaÌval a 3.) leÌpeÌsben.
    - Ha a veÌgrehajtandoÌ egyseÌg az Fi felteÌtel (1 â¤ i â¤ n), akkor eÌrteÌkeljuÌk ki. Ha Fi igaz volt, akkor hajtsuk veÌgre a Ki muÌveletet, eÌs fejezzuÌk be a vezeÌrleÌst. KuÌloÌnben a veÌgrehajtaÌs az Mi muÌvelettel folytatoÌdik a 2.) leÌpeÌsben.
- A kezdoÌ- eÌs veÌgfelteÌteles ismeÌtleÌses vezeÌrleÌsek speciaÌlis esetei a hurok ismeÌtleÌses vezeÌrleÌsnek.
- A C nyelvben a ciklusmag folyamatos veÌgrehajtaÌsaÌnak megszakiÌtaÌsaÌra keÌt utasiÌtaÌs hasznaÌlhatoÌ:
- break: MegszakiÌtja a ciklust, a program veÌgrehajtaÌsa a ciklusmag utaÌni elsoÌ utasiÌtaÌssal folytatoÌdik. HasznaÌlhatoÌ a switch utasiÌtaÌsban is, hataÌsaÌra a program veÌgrehajtaÌsa a switch utaÌni elsoÌ utasiÌtaÌssal folytatoÌdik.
- continue: MegszakiÌtja a ciklusmag aktuaÌlis lefutaÌsaÌt, a vezeÌrleÌs a ciklus felteÌteleÌnek kieÌrteÌkeleÌseÌvel (while, do while) illetve az inkrementaÌloÌ kifejezeÌs kieÌrteÌkeleÌseÌvel (for) folytatoÌdik.

DiszkrÃ©t ismÃ©tlÃ©ses vezÃ©rlÃ©s:

DiszkreÌt ismeÌtleÌses vezeÌrleÌsroÌl akkor beszeÌluÌnk, ha a ciklusmagot veÌgre kell hajtani egy halmaz minden elemeÌre tetszoÌleges sorrendben. - Legyen x egy T tiÌpusuÌ vaÌltozoÌ, H a T eÌrteÌkkeÌszleteÌnek reÌszhalmaza, M pedig tetszoÌleges muÌvelet, amelynek nincs hataÌsa x eÌs H eÌrteÌkeÌre. A H halmazboÌl, x ciklusvaÌltozoÌboÌl eÌs M muÌveletboÌl (ciklusmagboÌl) keÌpzett diszkreÌt ismeÌtleÌses vezeÌrleÌs az alaÌbbi vezeÌrleÌsi eloÌiÌraÌst jelenti: - Ha a H halmaz minden elemeÌre veÌgrehajtottuk az M muÌveletet, akkor veÌge a vezeÌrleÌsnek. - EgyeÌbkeÌnt vegyuÌk a H halmaz egy olyan tetszoÌleges e elemeÌt, amelyre meÌg nem hajtottuk veÌgre az M muÌveletet, eÌs folytassuk a 3.) leÌpeÌssel. - Legyen x = e eÌs hajtsuk veÌgre az M muÌveletet, majd folytassuk az 1.) leÌpeÌssel. - A H halmaz szaÌmossaÌga hataÌrozza meg, hogy az M muÌvelet haÌnyszor hajtoÌdik veÌgre. Ha a H az uÌres halmaz, akkor a diszkreÌt ismeÌtleÌses vezeÌrleÌs az M muÌvelet veÌgrehajtaÌsa neÌlkuÌl befejezoÌdik. - A diszkreÌt ismeÌtleÌses vezeÌrleÌsnek nincs koÌzvetlen megvaloÌsiÌtaÌsa a C nyelvben.

6. EgyszerÅ± adattÃpusok: egÃ©sz, valÃ³s, logikai Ã©s karakter tÃpusok Ã©s kifejezÃ©sek. Az egyszerÅ± tÃpusok

reprezentÃ¡ciÃ³ja, szÃ¡mÃ¡brÃ¡zolÃ¡si tartomÃ¡nyuk, pontossÃ¡guk, memÃ³ria igÃ©nyÃ¼k Ã©s mÅ±veleteik. Az Ã¶sszetett adattÃpusok Ã©s a tÃpuskÃ©pzÃ©sek, valamint megvalÃ³sÃtÃ¡suk C nyelven. A pointer, a tÃ¶mb, a rekord Ã©s az uniÃ³ tÃpus. Az egyes tÃpusok szerepe, hasznÃ¡lata

EgyszerÅ± adattÃpusok: egÃ©sz, valÃ³s, logikai Ã©s karakter tÃpusok Ã©s kifejezÃ©sek. Az egyszerÅ± tÃpusok reprezentÃ¡ciÃ³ja, szÃ¡mÃ¡brÃ¡zolÃ¡si tartomÃ¡nyuk, pontossÃ¡guk, memÃ³ria igÃ©nyÃ¼k Ã©s mÅ±veleteik

Az elemi adattÃpusok Ã©rtÃ©keit nem lehet Ã¶nmagukban Ã©rtelmes rÃ©szekre bontani.

Ha a nyelv szintaktikÃ¡ja szerint a program egy adott pontjÃ¡n tÃpusnak kellene kÃ¶vetkeznie de az hiÃ¡nyzik, a fordÃtÃ³ a tÃpus helyÃ©re automatikusan int-et helyettesÃt.

EgÃ©sz tÃpusok

A C nyelvben az egÃ©sz tÃpus az int.

Az int tÃpus Ã©rtÃ©kkÃ©szlete az alÃ¡bbi kulcsszavakkal mÃ³dosÃthatÃ³:

- signed: A tÃpus elÅjeles Ã©rtÃ©keket fog tartalmazni (int, char).
- unsigned: A tÃpus csak elÅjeltelen, nemnegatÃv Ã©rtÃ©keket fog tartalmazni (int, char).
- short: RÃ¶videbb helyen tÃ¡rolÃ³dik, Ãgy kisebb lesz az Ã©rtÃ©kkÃ©szlet (int).
- long: Hosszabb helyen tÃ¡rolÃ³dik, Ãgy bÅvebb lesz az Ã©rtÃ©kkÃ©szlet (int). DuplÃ¡n is alkalmazhatÃ³ (long long).

Az egÃ©sz tÃpusok az Ã©rtÃ©kkÃ©szlet hatÃ¡rain belÃ¼li minden egÃ©sz Ã©rtÃ©ket pontosan Ã¡brÃ¡zolnak.

Az egyes gÃ©peken az egyes tÃpusok mÃ©rete mÃ¡s-mÃ¡s lehet, de minden C megvalÃ³sÃtÃ¡sban teljesÃ¼lnie kell a sizeof(short) â¤ sizeof(int) â¤ sizeof(long) â¤ sizeof(long long) relÃ¡ciÃ³nak.

A C nyelv kÃ¼lÃ¶nfÃ©le egÃ©sz adattÃpusai az Ã©rtÃ©khalmazukban kÃ¼lÃ¶nbÃ¶znek egymÃ¡stÃ³l, az Ã©rtelmezett mÅ±veletÃ¼kben megegyeznek

Az egÃ©sz adattÃpusokon Ã¡ltalÃ¡ban az 5 matematikai alapmÅ±veletet Ã©s az Ã©rtÃ©kadÃ¡s mÅ±veletÃ©t Ã©rtelmezzÃ¼k, de C nyelven ennÃ©l jÃ³val tÃ¶bbet.

Ãrtékadó mÅ±velet jobb oldalÃ¡n Ã¡llÃ³ kifejezÃ©s kiÃ©rtÃ©kelÃ©se fÃ¼ggetlen attÃ³l, hogy a bal oldalon milyen tÃpusÃº vÃ¡ltozÃ³ van.

A / mÅ±velet kÃ©t egÃ©sz Ã©rtÃ©kre alkalmazva maradÃ©kos osztÃ¡st jelent!

TÃ¡rolÃ¡s:

n bites tÃ¡rterÃ¼letnek 2^n Ã¡llapota van, vagyis egy n biten tÃ¡rolt adattÃpusnak legfeljebb ennyi kÃ¼lÃ¶nbÃ¶zÅ Ã©rtÃ©ke lehet.

EgÃ©sz tÃpusoknÃ¡l a kettes komplemenst szokÃ¡s hasznÃ¡lni, ha negatÃv Ã©rtÃ©kek is szerepelhetnek az Ã©rtÃ©khalmazban.

Kettes komplemens:

- van egy pozitÃv szÃ¡munk, Ã©s annak keressÃ¼k a negatÃv pÃ¡rjÃ¡t
- a szÃ¡mot kettes szÃ¡mrendszerben felÃrjuk
- invertÃ¡ljuk az Ã¶sszes bitet
- majd hozzÃ¡adunk a vÃ©gÃ©n egyet
- a kapott szÃ¡m lesz a szÃ¡m ellentettje

ÃrtÃ©khalmaz mÃ©rete:

Ha negatÃv szÃ¡mok nem szerepelnek az Ã©rtÃ©khalmazban, akkor az Ã©rtÃ©khalmaz a [0 ... 2^n â 1] zÃ¡rt intervallum. Ha az Ã©rtÃ©khalmazban negatÃv szÃ¡mok is szerepelnek, akkor az Ã©rtÃ©khalmaz a [â2^(nâ1) ... 2^(nâ1) â 1] zÃ¡rt intervallum.

MÅ±veletei:

- bitenkÃ©nti
    - negÃ¡ciÃ³
    - Ã©s
    - vagy
    - kizÃ¡rÃ³ vagy
    - balra lÃ©ptetÃ©s
    - jobbra lÃ©ptetÃ©s

Karakter tÃpus

A char adattÃpus a C nyelv eleve definiÃ¡lt elemi adattÃpusa, Ã©rtÃ©kkÃ©szlete 256 elemet tartalmaz.

A char adattÃpus egÃ©szkÃ©nt is hasznÃ¡lhatÃ³, de alapvetÅen karakterek (betÅ±k, szÃ¡mjegyek, ÃrÃ¡sjelek) tÃ¡rolÃ¡sÃ¡ra valÃ³.

- Hogy melyik Ã©rtÃ©khez melyik karakter tartozik, az az alkalmazott kÃ³dtÃ¡blÃ¡zattÃ³l fÃ¼gg.
- Bizonyos karakterek (Ã¡ltalÃ¡ban a rendezÃ©s szerint elsÅ nÃ©hÃ¡ny) vezÃ©rlÅ karakternek szÃ¡mÃtanak, Ã©s nem megjelenÃ-thetÅk.

Egy C programban karakter Ã©rtÃ©keket megadhatunk

- karakterkÃ³ddal szÃ¡mÃ©rtÃ©kkÃ©nt, vagy
- aposztrÃ³fok kÃ¶zÃ© Ãrt karakterrel

A speciÃ¡lis karaktereket, illetve magÃ¡t az aposztrÃ³fot (Ã©s vÃ©gsÅ soron tetszÅleges karaktert is) escape-szekvenciÃ¡kkal lehet megadni. Az escape-szekvenciÃ¡kat a \ (backslash) karakterrel kell kezdeni.

KonvertÃ¡ljunk egy tetszÅleges szÃ¡mjegy karaktert (ch) a neki megfelelÅ egÃ©sz szÃ¡mmÃ¡ Ã©s egy egyjegyÅ± egÃ©szet (i) karakterrÃ©:

```
i = ch - '0'; ch = i + '0';
```

ValÃ³s tÃpusok

A C nyelvben a valÃ³s adattÃpusok a float Ã©s double.

A double adattÃpus az alÃ¡bbi kulcsszÃ³val mÃ³dosÃthatÃ³: - long: ImplementÃ¡ciÃ³fÃ¼ggÅ mÃ³don 64, 80, 96 vagy 128 bites pontossÃ¡got megvalÃ³sÃtÃ³ adattÃpus

A valÃ³s adattÃpusok az Ã©rtÃ©kkÃ©szlet hatÃ¡rain belÃ¼l sem kÃ©pesek minden valÃ³s Ã©rtÃ©ket pontosan Ã¡brÃ¡zolni. Viszont az Ã©rtÃ©kkÃ©szlet hatÃ¡rain belÃ¼li minden a valÃ³s Ã©rtÃ©ket kÃ©pesek egy tÃpusfÃ¼ggÅ e relatÃv pontossÃ¡ggal Ã¡brÃ¡zolni, az a-hoz legkÃ¶zelebbi a tÃpus Ã¡ltal pontosan Ã¡brÃ¡zolhatÃ³ x valÃ³s Ã©rtÃ©kkel.

- A C nyelv kÃ¼lÃ¶nfÃ©le valÃ³s adattÃpusai az Ã©rtÃ©khalmazukban kÃ¼lÃ¶nbÃ¶znek egymÃ¡stÃ³l, az Ã©rtelmezett mÅ±veletÃ¼kben megegyeznek.
- ValÃ³s kifejezÃ©sben bÃ¡rmely valÃ³s vagy egÃ©sz tÃpusÃº tÃ©nyezÅ (akÃ¡r vegyesen tÃ¶bbfÃ©le is) szerepelhet.
- ValÃ³s konstans tÃpusa double, vagy a szÃ¡mleÃrÃ¡sban megadott tÃpus (f, l suffix).
- ÃrtÃ©kadÃ³ mÅ±velet jobb oldalÃ¡n Ã¡llÃ³ kifejezÃ©s kiÃ©rtÃ©kelÃ©se fÃ¼ggetlen attÃ³l, hogy a bal oldalon milyen tÃpusÃº vÃ¡ltozÃ³ van.
- A tÃpus pontatlansÃ¡ga miatt az == mÅ±veletet nagyon kÃ¶rÃ¼ltekintÅen kell hasznÃ¡lni!

ÃbrÃ¡zolÃ¡sa:

Egy valÃ³s Ã©rtÃ©ket tÃ¡rolÃ³ memÃ³riaterÃ¼let hÃ¡rom rÃ©szre oszthatÃ³: az elÅjelbitet, a tÃ¶rtet Ã©s az exponenciÃ¡lis kitevÅt kÃ³dolÃ³ rÃ©szre.

- Az elÅjelbit 0 Ã©rtÃ©ke a pozitÃv, 1 Ã©rtÃ©ke a negatÃv szÃ¡mokat jelÃ¶li
- A szÃ¡mot kettes szÃ¡mrendszerben 1.m Ã 2^k alakra hozzuk, majd az m szÃ¡mjegyeit eltÃ¡roljuk a tÃ¶rtnek, a k-nak egy tÃpusfÃ¼ggÅ b konstanssal nÃ¶velt Ã©rtÃ©kÃ©t pedig a kitevÅnek fenntartott rÃ©szen.
- Ãgy a tÃ¶rt rÃ©sz hossza az Ã¡brÃ¡zolÃ¡s pontossÃ¡gÃ¡t (az Ã©rtÃ©kes szÃ¡mjegyek szÃ¡mÃ¡t), a kitevÅ pedig az Ã©rtÃ©ktartomÃ¡ny mÃ©retÃ©t hatÃ¡rozza meg.
- Nagyon kicsi szÃ¡mokat speciÃ¡lisan 0.m Ã 2^(1âb) alakban tÃ¡rolhatunk, ekkor a kitevÅ Ã¶sszes bitje 0.
- Ha a kitevÅ Ã¶sszes bitje 1, az csupa 0 bitbÅl Ã¡llÃ³ tÃ¶rt esetÃ©n a â, minden mÃ¡s esetben NaN Ã©rtÃ©ket jelenti.
- A 32/64 bites float/double az 1 elÅjelbit mÃ¶gÃ¶tt 8/11 biten a kitevÅ b = 127-tel/1023-mal nÃ¶velt Ã©rtÃ©kÃ©t, majd 23/52 biten a tÃ¶rtet tÃ¡rolja.

Logikai tÃpus

A C nyelvnek csak a C99 szabvÃ¡ny Ã³ta rÃ©sze a logikai (_Bool) tÃpus (melynek Ã©rtÃ©kkÃ©szlete a {0, 1} halmaz), de azÃ©rt logikai Ã©rtÃ©kek persze elÅtte is keletkeztek.

A mÅ±veletek eredmÃ©nyekÃ©nt keletkezÅ logikai hamis Ã©rtÃ©ket a 0 egÃ©sz Ã©rtÃ©k reprezentÃ¡lja, Ã©s a 0 egÃ©sz Ã©rtÃ©k logikai Ã©rtÃ©kkÃ©nt Ã©rtelmezve hamisat jelent.

A mÅ±veletek eredmÃ©nyekÃ©nt keletkezÅ logikai igaz Ã©rtÃ©ket az 1 egÃ©sz Ã©rtÃ©k reprezentÃ¡lja, de bÃ¡rmely 0-tÃ³l kÃ¼lÃ¶nbÃ¶zÅ egÃ©sz Ã©rtÃ©k logikai Ã©rtÃ©kkÃ©nt Ã©rtelmezve igazat jelent.

stdbool.h-ban definiÃ¡lt a bool tÃpus Ã©s a true, false konstansok

KonstanskÃ©nt is definiÃ¡lhatjuk, pl

```

# define TRUE 1

# define FALSE 0

```

Az Ã¶sszetett adattÃpusok Ã©s a tÃpuskÃ©pzÃ©sek, valamint megvalÃ³sÃtÃ¡suk C nyelven. A pointer, a tÃ¶mb, a rekord Ã©s az uniÃ³ tÃpus. Az egyes tÃpusok szerepe, hasznÃ¡lata

Ãsszetett adattÃpusok, tÃpuskÃ©pzÃ©sek

Az Ã¶sszetett adattÃpusok Ã©rtÃ©kei tovÃ¡bb bonthatÃ³ak, tovÃ¡bbi Ã©rtelmezÃ©sÃ¼k lehetsÃ©ges.

A C nyelv Ã¶sszetett adattÃpusai:

- Pointer tÃpus
    - FÃ¼ggvÃ©ny tÃpus
- TÃ¶mb tÃpus
    - Sztringek
- Rekord tÃpus
    - Szorzat-rekord
    - EgyesÃtÃ©si-rekord

Pointer tÃpus

Az eddigi tÃ¡rgyalÃ¡sunkban szerepelt vÃ¡ltozÃ³k statikusak abban az Ã©rtelemben, hogy lÃ©tezÃ©sÃ¼k annak a blokknak a vÃ©grehajtÃ¡sÃ¡hoz kÃ¶tÃ¶tt, amelyben a vÃ¡ltozÃ³ deklarÃ¡lva lett. A programozÃ³nak a deklarÃ¡ciÃ³ helyÃ©n tÃºl nincs befolyÃ¡sa a vÃ¡ltozÃ³ lÃ©tesÃtÃ©sÃ©re Ã©s megszÃ¼ntetÃ©sÃ©re.

Az olyan vÃ¡ltozÃ³kat, amelyek a blokkok aktivizÃ¡lÃ¡sÃ¡tÃ³l fÃ¼ggetlenÃ¼l lÃ©tesÃthetÅk Ã©s megszÃ¼ntethetÅk, dinamikus vÃ¡ltozÃ³knak nevezzÃ¼k.

Dinamikus vÃ¡ltozÃ³k megvalÃ³sÃtÃ¡sÃ¡nak Ã¡ltalÃ¡nos eszkÃ¶ze a pointer tÃpus.

Egy pointer tÃpusÃº vÃ¡ltozÃ³ Ã©rtÃ©ke (elsÅ megkÃ¶zelÃtÃ©sben) egy meghatÃ¡rozott tÃpusÃº dinamikus vÃ¡ltozÃ³.

Pointer tÃpusÃº vÃ¡ltozÃ³t a * segÃtsÃ©gÃ©vel deklarÃ¡lhatunk:

```
tÃpus * vÃ¡ltozÃ³nÃ©v;
```

Az eddigiek sorÃ¡n lÃ©nyegÃ©ben azonosÃtottuk a vÃ¡ltozÃ³hivatkozÃ¡st Ã©s a hivatkozott vÃ¡ltozÃ³t.

A dinamikus vÃ¡ltozÃ³k megÃ©rtÃ©sÃ©hez viszont vilÃ¡gosan kÃ¼lÃ¶nbsÃ©get kell tennÃ¼nk az alÃ¡bbi hÃ¡rom fogalom kÃ¶zÃ¶tt:

- VÃ¡ltozÃ³hivatkozÃ¡s
- Hivatkozott vÃ¡ltozÃ³
- VÃ¡ltozÃ³ Ã©rtÃ©ke

A vÃ¡ltozÃ³hivatkozÃ¡s szintaktikus egysÃ©g, meghatÃ¡rozott formai szabÃ¡lyok szerint kÃ©pzett jelsorozat egy adott programnyelven, tehÃ¡t egy kÃ³drÃ©szlet.

A vÃ¡ltozÃ³ a program futÃ¡sa sorÃ¡n a program Ã¡ltal lefoglalt memÃ³riaterÃ¼let egy rÃ©sze, amelyen egy adott (elemi vagy Ã¶sszetett) tÃpusÃº Ã©rtÃ©k tÃ¡rolÃ³dik.

KÃ¼lÃ¶nbÃ¶zÅ vÃ¡ltozÃ³hivatkozÃ¡sok hivatkozhatnak ugyanarra a vÃ¡ltozÃ³ra, illetve ugyanaz a vÃ¡ltozÃ³hivatkozÃ¡s a vÃ©grehajtÃ¡s kÃ¼lÃ¶nbÃ¶zÅ idÅpontjaiban kÃ¼lÃ¶nbÃ¶zÅ vÃ¡ltozÃ³kra hivatkozhat.

Egy vÃ¡ltozÃ³hivatkozÃ¡shoz nem biztos, hogy egy adott idÅben tartozik hivatkozott vÃ¡ltozÃ³.

MÅ±veletek:

- NULL
    - NULL, nem tartozik hozzÃ¡ dinamikus vÃ¡ltozÃ³
- LÃ©tesÃt
    - `x = malloc(sizeof(E))`
- ÃrtÃ©kadÃ¡s
    - `x = y`
- TÃ¶rlÃ©s
    - `free(x)`
- Dereferencia: A pointer Ã¡ltal mutatott dinamikus vÃ¡ltozÃ³ elÃ©rÃ©se, eredmÃ©nye egy vÃ¡ltozÃ³hivatkozÃ¡s.
    - `*x`
- EgyenlÅ
    - `p == q`
- NemEgyenlÅ
    - `p != q`

A memÃ³riamÅ±veletekhez szÃ¼ksÃ©g van az stdlib.h vagy a memory.h hasznÃ¡latÃ¡ra.

malloc(S), lefoglal egy S mÃ©retÅ± memÃ³riaterÃ¼letet sizeof(E), megmondja, hogy egy E tÃpusÃº Ã©rtÃ©k mekkora helyet igÃ©nyel a memÃ³riÃ¡ban malloc(sizeof(E)), lÃ©trehoz egy E tÃpusÃº Ã©rtÃ©k tÃ¡rolÃ¡sÃ¡ra is alkalmas vÃ¡ltozÃ³ free(p), felszabadÃtja a p-hez tartozÃ³ memÃ³riaterÃ¼letet, ezutÃ¡n a p-hez nem lesz Ã©rvÃ©nyes vÃ¡ltozÃ³hivatkozÃ¡s

Linux alatt logikailag minden programnak sajÃ¡t memÃ³ria-tartomÃ¡nya van, amin belÃ¼l az egyes memÃ³riacÃmeket egy sorszÃ¡m azonosÃtja.

Pointer tÃpusÃº vÃ¡ltozÃ³ 32 bites rendszereken 4 bÃ¡jt, 64 bites rendszereken 8 bÃ¡jt hosszban a hozzÃ¡ tartozÃ³ dinamikus vÃ¡ltozÃ³hoz foglalt memÃ³riamezÅ kezdÅcÃmÃ©t (sorszÃ¡mÃ¡t) tartalmazza.

A pointer Ã©rtÃ©ke tehÃ¡t (mÃ¡sodik megkÃ¶zelÃtÃ©sben) Ã©rtelmezhetÅ egy tetszÅleges memÃ³riacÃmkÃ©nt is, amely Ã©rtelmezÃ©s egybeesik a pointer megvalÃ³sÃtÃ¡sÃ¡val.

Ilyen mÃ³don viszont Ã©rtelmezhetjÃ¼k a cÃmkÃ©pzÅ mÅ±veletet, ami egy vÃ¡ltozÃ³ memÃ³riabeli pozÃciÃ³jÃ¡t, cÃmÃ©t adja vissza.

- CÃm
    - `p = &x`

A void* egy speciÃ¡lis, Ãºgynevezett tÃpustalan pointer. Az ilyen tÃpusÃº pointerek âcsakâ memÃ³riacÃmek tÃ¡rolÃ¡sÃ¡ra alkalmasak, a dereferencia mÅ±velet alkalmazÃ¡sa rÃ¡juk Ã©rtelmetlen. Viszont minden tÃpusÃº pointerrel kompatibilisek Ã©rtÃ©kadÃ¡s Ã©s Ã¶sszehasonlÃtÃ¡s tekintetÃ©ben.

TÃ¶mb tÃpus

Algoritmusok tervezÃ©sekor gyakran elÅfordul, hogy adatok sorozatÃ¡val kell dolgozni, vagy mert az input adatok sorozatot alkotnak, vagy mert a feladat megoldÃ¡sÃ¡hoz kell.

TegyÃ¼k fel, hogy a sorozat rÃ¶gzÃtett elemszÃ¡mÃº (n) Ã©s mindegyik komponensÃ¼k egy megadott (elemi vagy Ã¶sszetett) tÃpusbÃ³l (E) valÃ³ Ã©rtÃ©k.

Ekkor tehÃ¡t egy olyan Ã¶sszetett adathalmazzal van dolgunk, amelynek egy eleme A = (a 0 , . . . , a nâ1), ahol a i â E , âi â (0, . . . , n â 1)-re.

Ha az ilyen sorozatokon a kÃ¶vetkezÅ mÅ±veleteket Ã©rtelmezzÃ¼k, akkor egy (absztrakt) adattÃpushoz jutunk, amit TÃ¶mb tÃpusnak nevezÃ¼nk.

JelÃ¶ljÃ¼k ezt a TÃ¶mb tÃpust T -vel, a 0, . . . , n â 1 intervallumot pedig I-vel.

MÅ±veletek

- Kiolvas
    - a sorozat i. elemÃ©nek kiolvasÃ¡sa egy vÃ¡ltozÃ³ba
- MÃ³dosÃt
    - a sorozat i. elemÃ©nek mÃ³dosÃtÃ¡sa egy E tÃpusÃº Ã©rtÃ©kre
- ÃrtÃ©kadÃ¡s
    - a vÃ¡ltozÃ³ felveszi a tÃ¶mb Ã©rtÃ©kÃ©t

TÃ¶mb tÃpusÃº vÃ¡ltozÃ³t az alÃ¡bbi mÃ³don deklarÃ¡lhatunk:

```
tÃpus vÃ¡ltozÃ³nÃ©v[elemszÃ¡m];
```

TÃ¶mbelem hivatkozÃ¡sra a `[]` operÃ¡tort hasznÃ¡ljuk.

Ez egy olyan tÃ¶mbÃ¶kÃ¶n Ã©rtelmezett mÅ±velet C-ben, ami nagyon magas precedenciÃ¡val rendelkezik Ã©s balasszociatÃv.

Egy tÃ¶mbre a tÃ¶mbindexelÃ©s operÃ¡tort (megfelelÅ index hasznÃ¡latÃ¡val) alkalmazva a tÃ¶mb adott elemÃ©t vÃ¡ltozÃ³kÃ©nt kapjuk vissza.

Rekord tÃpus

A tÃ¶mb tÃpus nagyszÃ¡mÃº, de ugyanazon tÃpusÃº adat tÃ¡rolÃ¡sÃ¡ra alkalmas.

ProblÃ©mÃ¡k megoldÃ¡sa sorÃ¡n viszont gyakran elÅfordul, hogy kÃ¼lÃ¶nbÃ¶zÅ tÃpusÃº, de logikailag Ã¶sszetartozÃ³ adatelemek egyÃ¼ttesÃ©vel kell dolgozni.

Az ilyen adatok tÃ¡rolÃ¡sÃ¡ra szolgÃ¡lnak a rekord tÃpusok, ezek lÃ©trehozÃ¡sÃ¡ra pedig a rekord tÃpuskÃ©pzÃ©sek.

Ha az egyes tÃpusÃº adatokat egyszerre kell tudnunk tÃ¡rolni, szorzat-rekordrÃ³l beszÃ©lÃ¼nk.

Az Ãºj adattÃpusra a T=Rekord(T 1 , . . . , T k) jelÃ¶lÃ©st hasznÃ¡ljuk Ã©s szorzat-rekordnak vagy struktÃºrÃ¡nak nevezzÃ¼k.

- kiolvas
- mÃ³dosÃt
- Ã©rtÃ©kadÃ¡s

```
typedef struct T { T1 M1; ... Tk Mk; } T;
```

A fenti tÃpuskÃ©pzÃ©sben az M1,. . . ,Mk azonosÃtÃ³kat mezÅazonosÃtÃ³knak (tagnak, member-nek) hÃvjuk Ã©s lokÃ¡lisak a tÃpuskÃ©pzÃ©sre nÃ©zve.

Az absztrakt tÃpus mÅ±veletei mezÅhivatkozÃ¡sok segÃtsÃ©gÃ©vel valÃ³sÃthatÃ³ak meg.

A mezÅhivatkozÃ¡sra a . operÃ¡tort hasznÃ¡ljuk. Ez egy olyan rekordokon Ã©rtelmezett mÅ±velet C-ben, ami nagyon magas precedenciÃ¡val rendelkezik Ã©s balasszociatÃv.

Egy rekordra a mezÅkivÃ¡lasztÃ¡s operÃ¡tort (megfelelÅ mezÅnÃ©vvel) alkalmazva a rekord mezÅjÃ©t vÃ¡ltozÃ³kÃ©nt kapjuk vissza.

UniÃ³ tÃpus

Ha az egyes tÃpusÃº adatokat nem kell egyszerre tÃ¡rolni, egyesÃtett-rekordrÃ³l beszÃ©lÃ¼nk

A T halmazon is a szorzat rekordhoz hasonlÃ³ mÃ³don Ã©rtelmezhetÃ¼nk kiolvasÃ³ Ã©s mÃ³dosÃtÃ³ mÅ±veletet.

Az Ãºj adattÃpust a T 0 vÃ¡ltozati tÃpusbÃ³l Ã©s T 1 , . . . , T k egyesÃtÃ©si-tag tÃpusokbÃ³l kÃ©pzett egyesÃtett-rekord tÃpusnak nevezzÃ¼k.

```
typedef union T { T1 M1; ... Tk Mk; } T;
```

A union tÃpusÃº vÃ¡ltozÃ³ szÃ¡mÃ¡ra foglalt memÃ³ria mÃ©rete, amely a sizeof fÃ¼ggvÃ©nnyel lekÃ©rdezhetÅ: `sizeof(T) = max{sizeof(T1), ..., sizeof(Tk)}`

Valamennyi vÃ¡ltozati mezÅ ugyanazon a memÃ³riacÃmen kezdÅdik, ami megegyezik a teljes union tÃpusÃº Ã©rtÃ©k cÃmÃ©vel (azaz minden mezÅ eltolÃ¡sa, offset-je 0).

7. Objektum orientÃ¡lt paradigma Ã©s annak megvalÃ³sÃtÃ¡sa a JAVA Ã©s C++ nyelvekben. Az absztrakt adattÃpus, az osztÃ¡ly. Az egysÃ©gbe zÃ¡rÃ¡s, az informÃ¡ciÃ³ elrejtÃ©s, az Ã¶rÃ¶klÅdÃ©s, az Ãºjrafelhasznï¿½lÃ¡s Ã©s a polimorfizmus. A polimorfizmus feloldÃ¡sÃ¡nak mÃ³dszere

Objektum orientÃ¡lt paradigma

Az objektum orientÃ¡l paradigma az objektumok fogalmÃ¡n alapulÃ³ programozÃ¡si paradigma. Az objektumok egysÃ©gbe foglaljÃ¡k az adatokat Ã©s a hozzÃ¡juk tartozÃ³ mÅ±veleteket. A program egymÃ¡ssal kommunikÃ¡lÃ³ objektumok Ã¶sszessÃ©gÃ©bÅl Ã¡ll melyek hasznÃ¡ljÃ¡k egymÃ¡s mÅ±veleteit Ã©s adatait.

Az objektum-orientÃ¡ltsÃ¡g hÃ¡rom alapillÃ©re:

- EgysÃ©gbezÃ¡rÃ¡s Ã©s adatelrejtÃ©s (Encapsulation & information hiding)
- Ãjrafelhasználás, polimorfizmus Ã©s Ã¶rÃ¶klÅdÃ©s (Reusability, polymorphism & inheritence)
- Magasabb fokÃº absztrakciÃ³

EgysÃ©gbezÃ¡rÃ¡s Ã©s adatelrejtÃ©s

Az egysÃ©gbe zÃ¡rÃ¡s azt fejezi ki, hogy az Ã¶sszetartozÃ³ adatok Ã©s fÃ¼ggvÃ©nyek, eljÃ¡rÃ¡sok egyÃ¼tt vannak, egy egysÃ©gbe tartoznak. TovÃ¡bbi fontos fogalom az adatelrejtÃ©s, ami azt jelenti, hogy kÃvÃ¼lrÅl csak az fÃ©rhetÅ hozzÃ¡ kÃ¶zvetlenÃ¼l, amit az objektum osztÃ¡lya megenged.

Ha az objektum, illetve osztÃ¡ly elrejti az Ã¶sszes adattagjÃ¡t, Ã©s csak bizonyos metÃ³dusokon keresztÃ¼l fÃ©rhetnek hozzÃ¡ a kliensek, akkor az egysÃ©gbe zÃ¡rÃ¡s az absztrakciÃ³t Ã©s informÃ¡ciÃ³elrejtÃ©s erÅs formÃ¡jÃ¡t valÃ³sÃtja meg

Az osztÃ¡ly Ã©s objektum

Absztrakt adattÃpus: Az adattÃpus leÃrÃ¡sÃ¡nak legmagasabb szintje, amelyben az adattÃpust Ãºgy specifikÃ¡ljuk, hogy az adatok Ã¡brÃ¡zolÃ¡sÃ¡ra Ã©s a mÅ±veletek implementÃ¡ciÃ³jÃ¡ra semmilyen elÅÃrÃ¡st nem adunk.

OsztÃ¡ly: Egy absztrakt adattÃpus. Az adattagokbÃ³l Ã©s a rajta elvÃ©gezhetÅ mÅ±veleteket zÃ¡rja egy egysÃ©gbe. EgÃ©szen konkrÃ©tan objektumok csoportjÃ¡nak leÃrÃ¡sa, amelyeknek kÃ¶zÃ¶s az attribÃºtumaik, operÃ¡ciÃ³ik Ã©s szemantikus viselkedÃ©sÃ¼k van. UgyanÃºgy viselkedik, mint minden egyÃ©b primitÃv tÃpus, tehÃ¡t pl. vÃ¡ltozÃ³ (objektum) hozhatÃ³ lÃ©tre belÅlÃ¼k.

- LÃ©trehozÃ¡s: Java-ban Ã©s C++-ban is a class kulcsszÃ³val tudunk osztÃ¡lyokat definiÃ¡lni. Az osztÃ¡lyokbÃ³l tetszÅleges mennyisÃ©gben lÃ©trehozhatunk pÃ©ldÃ¡nyokat, azaz objektumokat.

Objektum: Egy vÃ¡ltozÃ³, melynek tÃpusa valamely objektumosztÃ¡ly, vagyis az osztÃ¡ly egy pÃ©ldÃ¡nya amely rendelkezik Ã¡llapottal, viselkedÃ©ssel, identitÃ¡ssal. Az objektumok gyakran megfeleltethetÅk a valÃ³ Ã©let objektumainak vagy egyedeinek

- Ã¡llapot: Egy az objektum lehetsÃ©ges lÃ©tezÃ©si lehetÅsÃ©gei kÃ¶zÃ¼l (a tulajdonsÃ¡gok aktuÃ¡lis Ã©rtÃ©ke, pl: lÃ¡mpaBekapcsolva true vagy false)
- viselkedÃ©s: Az objektum viselkedÃ©se annak leÃrÃ¡sa, hogy az objektum hogy reagÃ¡l mÃ¡s objektumok kÃ©rÃ©seire. (metÃ³dusok, pl: lÃ¡mpa.bekapcsol())
- identitÃ¡s: Minden objektum egyedi, mÃ©g akkor is, ha Ã©ppen ugyanabban az Ã¡llapotban vannak, Ã©s ugyanolyan viselkedÃ©st kÃ©pesek megvalÃ³sÃtani.

InformÃ¡ciÃ³ elrejtÃ©se

A lÃ¡thatÃ³sÃ¡gok segÃtsÃ©gÃ©vel tudjuk szabÃ¡lyozni adattagok, metÃ³dusok elÃ©rÃ©sÃ©t, ugyanis ezeket az objektumorientÃ¡lt paradigma Ã©rtelmÃ©ben korlÃ¡tozni kell, kÃvÃ¼lrÅl csak Ã©s kizÃ¡rÃ³lag ellenÅrzÃ¶tt mÃ³don lehessen ezeket elÃ©rni, hasznÃ¡lni.

Az adattagok, Ã©s metÃ³dusok lÃ¡thatÃ³sÃ¡gÃ¡nak vezÃ©rlÃ©sÃ©hez vannak kulcsszavak, amelyekkel megfelelÅen el tudjuk rejteni Åket.

LÃ¡thatÃ³sÃ¡gi opciÃ³k

- public: mindenhonnan lÃ¡thatÃ³
- protected: csak az osztÃ¡ly scope-jÃ¡n belÃ¼l, illetve a kÃ©sÅbb az adott osztÃ¡lybÃ³l szÃ¡rmaztatott gyerekosztÃ¡lyokon belÃ¼l lehet hivatkozni.
- private: csak az adott osztÃ¡lyon belÃ¼l lehet hivatkozni rÃ¡

(Java-ban alapÃ©rtelmezetten package private (az adott packagen belÃ¼l public, egyÃ©bkÃ©nt private) a lÃ¡thatÃ³sÃ¡g, mÃg C++-ban private)

TÃ¶rekedni kell a minÃ©l nagyobb adatbiztonsÃ¡gra Ã©s informÃ¡ciÃ³ elrejtÃ©sre: az adat tagok lÃ¡thatÃ³sÃ¡ga legyen private, esetleg indokolt esetben protected.

ÃrÃ¶klÅdÃ©s

OsztÃ¡lyok kÃ¶zÃ¶tt Ã©rtelmezett viszony, amely segÃtsÃ©gÃ©vel egy Ã¡ltalÃ¡nosabb tÃpusbÃ³l (ÅsosztÃ¡ly) egy sajÃ¡tosabb tÃpust tudunk lÃ©trehozni (utÃ³dosztÃ¡ly). Az utÃ³dosztÃ¡ly adatokat Ã©s mÅ±veleteket Ã¶rÃ¶kÃ¶l, kiegÃ©szÃti ezeket sajÃ¡t adatokkal Ã©s mÅ±veletekkel, illetve felÃ¼lÃrhat bizonyos mÅ±veleteket. A kÃ³d ÃºjrafelhasznÃ¡lÃ¡sÃ¡nak egyik mÃ³dja. MegkÃ¼lÃ¶nbÃ¶ztetÃ¼nk egyszeres Ã©s tÃ¶bbszÃ¶rÃ¶s Ã¶rÃ¶kÃtÃ©st.

A hasonlÃ³sÃ¡g kifejezÃ©se az Ås felÃ© az Ã¡ltalÃ¡nosÃtÃ¡s. A kÃ¼lÃ¶nbsÃ©g a gyerek felÃ© a specializÃ¡lÃ¡s.

Java: az extends kulcsszÃ³val tudjuk jelezni, hogy az adott osztÃ¡ly egy mÃ¡sik osztÃ¡lynak a leszÃ¡rmazottja. Java-ban egyszeres Ã¶rÃ¶klÅdÃ©s van, vagyis egy osztÃ¡ly csak is egy ÅsosztÃ¡lybÃ³l szÃ¡rmazhat (viszont tÃ¶bb interfÃ©szt implementÃ¡lhat)

- super: segÃtsÃ©gÃ©vel gyerekosztÃ¡lybÃ³l hivatkozhatunk szÃ±lÅosztÃ¡ly adattagjaira Ã©s megÃ³tudaira.

C++: Az osztÃ¡ly neve utÃ¡n vesszÅvel elvÃ¡lasztva lehet megadni az ÅsosztÃ¡lyokat Ã©s velÃ¼k egyÃ¼tt a lÃ¡thatÃ³sÃ¡gaikat. LehetÅsÃ©g van tÃ¶bbszÃ¶rÃ¶s Ã¶rÃ¶klÅdÃ©sre is

- Az Ã¶rÃ¶klÅdÃ©s sorÃ¡n lehetÅsÃ©g van az Ås osztÃ¡ly tagjainak lÃ¡thatÃ³sÃ¡gi opciÃ³jÃ¡n vÃ¡ltoztatni. Ezt az Ås osztÃ¡lyok felsorolÃ¡sakor kell definiÃ¡lni. Az vÃ¡ltoztatÃ¡s csak szigorÃtÃ¡st (korlÃ¡tozÃ¡st) jelenthet. Az alÃ¡bbi tÃ¡blÃ¡zat a gyermek osztÃ¡lybeli lÃ¡thatÃ³sÃ¡got mutatja be az Ås osztÃ¡lybeli lÃ¡thatÃ³sÃ¡g Ã©s a mÃ³dosÃtÃ¡s fÃ¼ggvÃ©nyÃ©ben:

VirtuÃ¡lis Ã¶rÃ¶klÅdÃ©s

TÃ¶bbszÃ¶rÃ¶s Ã¶rÃ¶klÅdÃ©snÃ©l elÅfordulhat olyan eset, amikor egy-egy Ås osztÃ¡ly az Ã¶rÃ¶klÅdÃ©si hierarchia kÃ¼lÃ¶nbÃ¶zÅ pontjÃ¡n ismÃ©t megjelenik. Ekkor a gyermek osztÃ¡lyban ennek az Ås osztÃ¡lynak tÃ¶bb pÃ©ldÃ¡nya jelenhet meg. Erre nÃ©hÃ¡ny esetben nincs szÃ¼ksÃ©g, pÃ©ldÃ¡ul ha az Ås osztÃ¡ly csak egy eljÃ¡rÃ¡s-erÅforrÃ¡s, akkor minden esetben elegendÅ egyetlen elÅfordulÃ¡s a gyermek osztÃ¡lyokban.

A virtuÃ¡lis Ås osztÃ¡lyt az ÅrÃ¶klÅdÃ©snÃ©l az Ås osztÃ¡lyok felsorolÃ¡sakor virtual mÃ³dosÃtÃ³val kell jelezni.

(Ha nem adom meg a virtual mÃ³dosÃtÃ³ szÃ³t, akkor az A osztÃ¡ly tÃ¶bbszÃ¶r fog megjelenni a D osztÃ¡ly pÃ©ldÃ¡nyaiban. HivatkozÃ¡snÃ¡l mindig meg kell mondani, hogy az A melyik pÃ©ldÃ¡nyÃ¡rÃ³l van szÃ³: C::A::m_iN, B::A::m_iN.)

â

Ãjrafelhasználás, Polimorfizmus:

Az Ãºjrafelhasználhatósága az OOP egyik legfontosabb elÅnye.

Az a jelensÃ©g, hogy egy vÃ¡ltozÃ³ nem csak egyfajta tÃpusÃº objektumra hivatkozhat a polimorfizmus.

A polimorfizmus lehetÅvÃ© teszi szÃ¡munkra, hogy egyetlen mÅ±veletet kÃ¼lÃ¶nbÃ¶zÅ mÃ³don hajtsunk vÃ©gre. MÃ¡s szavakkal, a polimorfizmus lehetÅvÃ© teszi egy interfÃ©sz definiÃ¡lÃ¡sÃ¡t Ã©s tÃ¶bb megvalÃ³sÃtÃ¡st. Az objektumok felcserÃ©lhetÅsÃ©gÃ©t biztosÃ-tja. Az objektumok ÅstÃpusai alapjÃ¡n kezeljÃ¼k, Ãgy a kÃ³d nem fÃ¼gg a specifikus tÃpusoktÃ³l.

Polimorfizmusra kÃ©t lehetÅsÃ©g van:

- statikus polimorfizmus (korai hozzÃ¡rendelÃ©s) - a hÃvott metÃ³dus nevÃ©nek Ã©s cÃmÃ©nek Ã¶sszerendelÃ©se szerkesztÃ©skor tÃ¶rtÃ©nik meg. A futtathatÃ³ programban mÃ¡r fix metÃ³duscÃmek talÃ¡lhatÃ³k. (statikus, private, final metÃ³dusok)
- dinamikus polimorfizmus (kÃ©sÅi hozzÃ¡rendelÃ©s) - metÃ³dus nevÃ©nek Ã©s cÃmÃ©nek hozzÃ¡rendelÃ©se a hÃvÃ¡s elÅtti sorban tÃ¶rtÃ©nik, futÃ¡si idÅben

VirtuÃ¡lis eljÃ¡rÃ¡sok

Egy virtuÃ¡lis eljÃ¡rÃ¡s cÃmÃ©nek meghatÃ¡rozÃ¡sa indirekt mÃ³don, futÃ¡s kÃ¶zben tÃ¶rtÃ©nik.

Java-ban eleve csak virtuÃ¡lis eljÃ¡rÃ¡sok vannak (kivÃ©ve a final metÃ³dusokat, amelyeket nem lehet felÃ¼ldefiniÃ¡lni Ã©s a private metÃ³dusokat, amelyeket nem lehet Ã¶rÃ¶kÃ¶lni)

C++-ban a virtuÃ¡lis fÃ¼ggvÃ©nytÃ¡bla tartja nyilvÃ¡n a virtuÃ¡lis eljÃ¡rÃ¡sok cÃmeit. A VFT tÃ¡blÃ¡zat Ã¶rÃ¶klÅdik, feltÃ¶ltÃ©sÃ©rÅl a konstruktor gondoskodik. A szÃ¡rmaztatott osztÃ¡ly konstruktora mÃ³dosÃtja a virtuÃ¡lis fÃ¼ggvÃ©nytÃ¡blÃ¡t, kijavÃtja az ÅsosztÃ¡lybÃ³l Ã¶rÃ¶kÃ¶lt metÃ³duscÃmeket. Amikor a konstruÃ¡lÃ¡si folyamat vÃ©get Ã©r, a VFT tÃ¡blÃ¡zat minden sora Ã©rtÃ©ket kap, mÃ©gpedig a tÃ©nylegesen lÃ©trehozott osztÃ¡lynak megfelelÅ metÃ³dus cÃmeket. A VFT tÃ¡blÃ¡zat sorai ezutÃ¡n mÃ¡r nem vÃ¡ltoznak meg.

- VirtuÃ¡lis eljÃ¡rÃ¡sokat a virtual kulcsszÃ³val tudunk lÃ©trehozni. Az Ãºjrafelhasználás sorÃ¡n nagy valÃ³szÃnÅ±sÃ©ggel mÃ³dosÃ-tÃ¡sra kerÃ¼lÅ eljÃ¡rÃ¡sokat a szÃ¼lÅ osztÃ¡lyokban cÃ©lszerÅ± egybÅl virtuÃ¡lisra megÃrni, mert ezzel jelentÅs munkÃ¡t lehet megtakarÃtani a kÃ©sÅbbiekben.

Absztrakt osztÃ¡ly, interfÃ©sz

Java: Absztrakt osztÃ¡lyok

- Az abstract kulcsszÃ³val hozhatÃ³ lÃ©tre.
- Egy absztrakt osztÃ¡lybÃ³l nem hozhatÃ³ lÃ©tre objektum.
- Tartalmazhat absztrakt metÃ³dusokat (absztrakt metÃ³dusnak nincs implementÃ¡ciÃ³ja, azaz tÃ¶rzse), illetve nem absztraktokat

- Gyerek osztÃ¡lyban az abstract metÃ³dusokat felÃ¼l KELL definiÃ¡lni, ha pÃ©ldÃ¡nyosÃthatÃ³ osztÃ¡lyt szeretnÃ©nk
- Ha egy osztÃ¡ly rendelkezik legalÃ¡bb egy absztrakt metÃ³dussal, akkor osztÃ¡lynak is absztraktnak kell lennie
- Lehetnek adattagjai

InterfÃ©sz

- Az interface kulcsszÃ³val lehet lÃ©trehozni
- Egy speciÃ¡lis absztrakt osztÃ¡ly
- Nincsenek sem megvalÃ³sÃtott metÃ³dusok, sem adattagok. CsupÃ¡n metÃ³dus deklarÃ¡ciÃ³kat tartalmaz
- GyerekosztÃ¡lyban az implements kulcsszÃ³val lehet implementÃ¡lni

C++: Absztrakt osztÃ¡lyok:

A tÃ¶rzs nÃ©lkÃ¼li virtuÃ¡lis eljÃ¡rÃ¡sokat pure virtual eljÃ¡rÃ¡soknak nevezzÃ¼k (pl.: virtual int getArea() = 0;). A pure virtual eljÃ¡rÃ¡s egy Ã¼res (NULL) bejegyzÃ©st foglal el a VFT (Virtual Function Table) tÃ¡blÃ¡zatban. Ha egy osztÃ¡ly ilyen eljÃ¡rÃ¡st tartalmaz, akkor azt absztrakt osztÃ¡lynak nevezzÃ¼k amiatt, mert ebbÅl az osztÃ¡lybÃ³l objektum pÃ©ldÃ¡nyokat lÃ©trehozni nem lehet. A gyermek osztÃ¡lyokban minden pure virtual eljÃ¡rÃ¡st megfelelÅ tÃ¶rzzsel kell ellÃ¡tni, ezt a fordÃtÃ³ ellenÅrzi. AmÃg egyetlen pure virtual eljÃ¡rÃ¡s is marad, az osztÃ¡ly absztrakt lesz.

8. Objektumok Ã©letciklusa, lÃ©trehozÃ¡s, inicializÃ¡lÃ¡s, mÃ¡solÃ¡s, megszÃ¼ntetÃ©s. Dinamikus, lokÃ¡lis Ã©s statikus objektumok lÃ©trehozÃ¡sa. A statikus adattagok Ã©s metÃ³dusok, valamint szerepÃ¼k a programozÃ¡sban. OperÃ¡ciÃ³ Ã©s operÃ¡tor overloading a JAVA Ã©s C++ nyelvekben. KivÃ©telkezelÃ©s

Objektumok lÃ©trehozÃ¡sa

Az objektumokat Java-ban Ã©s C++-ban is tÃ¡rolhatjuk statikusan (az adatszegmensben), a veremben (lokÃ¡lisan) vagy a heapben (dinamikusan).

Java-ban az objektumok mindig a heap-ben keletkeznek, kivÃ©ve a primitÃv tÃpusokat. Az osztÃ¡lyok konstruktora fogja inicializÃ¡lni az objektumot. A konstruktor neve meg kell egyezzen az osztÃ¡ly nevÃ©vel. A konstruktornak nincs visszatÃ©rÃ©si Ã©rtÃ©ke, de paramÃ©terei lehetnek, amelyekkel meg lehet adni, hogy hogyan inicializÃ¡ljuk az objektumot.

A new operÃ¡tor: - Szintaxis: new OsztÃ¡ly(args) - LÃ©trehozÃ¡s lÃ©pÃ©sei: - Lefoglalja a szÃ¡mÃ¡ra szÃ¼ksÃ©ges memÃ³riÃ¡t - MeghÃvja az osztÃ¡ly konstruktorÃ¡t - Visszaadja az objektumra mutatÃ³ referenciÃ¡t

Egy osztÃ¡lyhoz kÃ©szÃthetÃ¼nk tÃ¶bb konstruktort, amelyek kÃ¼lÃ¶nbÃ¶zÅ paramÃ©terlistÃ¡val rendelkeznek.

C++-ban is hasonlÃ³an mÅ±kÃ¶dik a konstruktor: a konstruktor inicializÃ¡lja az objektumot, azaz tÃ¶lti fel az adattagjait Ã©rtÃ©kekkel, tÃ¶bb kÃ¼lÃ¶nbÃ¶zÅ paramÃ©ter listÃ¡jÃº konstruktort lehet lÃ©trehozni egy osztÃ¡lyhoz, a konstruktor neve meg kell egyezzen az osztÃ¡ly nevÃ©vel Ã©s visszaadott Ã©rtÃ©ke nem lehet.

A paramÃ©ter nÃ©lkÃ¼li konstruktor eljÃ¡rÃ¡s neve: alapÃ©rtelmezett (default) konstruktor. Csak Ås osztÃ¡lyokban kÃ¶telezÅ, akkor ha az osztÃ¡lybÃ³l gyermek osztÃ¡lyokat szeretnÃ©nek lÃ©trehozni Ã¶rÃ¶klÅdÃ©ssel. MegvalÃ³sÃthatÃ³ olymÃ³don is, hogy egy nem default konstruktor minden paramÃ©terÃ©hez default eljÃ¡rÃ¡s paramÃ©tereket adunk (pl. Osztaly(int x = 1, int y = 2)).

Amennyiben egy gyermek osztÃ¡ly konstruÃ¡lunk, akkor a konstruktor minden esetben meg kell hÃvja rekurzÃvan az Ås osztÃ¡ly(ok) konstruktorait mielÅtt elkezdenÃ© a sajÃ¡t eljÃ¡rÃ¡s tÃ¶rzsÃ©t vÃ©grehajtani. Java-ban ez impliciten megtÃ¶rtÃ©nik, ha az ÅsosztÃ¡lynak van default konstruktora.

C++-ban a heapbeli objektumok lÃ©trehozÃ¡sa a new operÃ¡torral tÃ¶rtÃ©nik, megszÃ¼ntetÃ©sÃ¼k pedig a delete operÃ¡torral. A lÃ©trehozÃ¡shoz nem elegendÅ a memÃ³ria megfelelÅ mÃ©retben tÃ¶rtÃ©nÅ lefoglalÃ¡sa, hanem a konstruktor eljÃ¡rÃ¡st is meg kell hÃvni. (EzÃ©rt nem lehet objektum pÃ©ldÃ¡nyt lÃ©trehozni malloc eljÃ¡rÃ¡ssal.) A new operÃ¡torral egyetlen objektum pÃ©ldÃ¡nyt vagy megadott mÃ©retÅ± tÃ¶mbÃ¶t hozhatunk lÃ©tre. A new operÃ¡tor alkalmazÃ¡sÃ¡nak eredmÃ©nye mindig egy pointer a new operandusÃ¡ban megadott osztÃ¡lyra.

Szintaxis:

TÃ¶mbÃ¶k foglalÃ¡sakor a default konstruktor hÃvÃ³dik meg. MegszÃ¼ntetÃ©sÃ¼knÃ©l az Ã¼res [] zÃ¡rÃ³jelpÃ¡r hasznÃ¡lata kÃ¶telezÅ.

C++ban az objektum megszÃ¼ntetÃ©se elÅtti takarÃtÃ¡st, erÅforrÃ¡s felszabadÃtÃ¡st a destruktor vÃ©gzi. A neve meg kell egyezzen az osztÃ¡ly nevÃ©vel, ami elÃ© egy ~ (tilde) jelet is kell tenni. ParamÃ©tere Ã©s visszaadott Ã©rtÃ©ke nem lehet. A destruktor mÃ¡r nem Ã¡llÃthatja meg az objektum megszÃ¼ntetÃ©sÃ©t. Amikor a destruktor vÃ©get Ã©r, az objektumot a rendszer a memÃ³riÃ¡bÃ³l tÃ¶rli. Mindig a gyerek osztÃ¡ly destruktora hÃvÃ³dik meg elÅszÃ¶r, Ã©s azt kÃ¶veti rekurzÃvan az Ås osztÃ¡lyok destruktorainak a meghÃvÃ¡sa.

Java-ban nincs szÃ¼ksÃ©g a heap-ben lÃ©trehozott objektumok manuÃ¡lis tÃ¶rlÃ©sÃ©re. A takarÃtÃ¡st automatikusan elvÃ©gzi a garbage collector (szemÃ©tgyÅ±jtÅ). Ez biztonsÃ¡gosabb, a programozÃ³nak nem kell emlÃ©keznie, hogy fel kell szabadÃtani az erÅforrÃ¡sokat, viszont sokkal lassabb. A szemÃ©tgyÅ±jtÃ©st kÃ©zzel is el lehet indÃtani, de ez nem egyenlÅ a destruktorral, kiszÃ¡mÃthatatlan, hogy mikor fog vÃ©grehajtÃ³dni.

Objektum mÃ¡solÃ¡s

Akkor beszÃ©lÃ¼nk klÃ³nozÃ¡srÃ³l, ha egy objektum pÃ©ldÃ¡nyt kÃ©t (vagy tÃ¶bb) pÃ©ldÃ¡nyban sokszorosÃtunk Ãºgy, hogy az egyes pÃ©ldÃ¡nyok adat tagjai azonosak lesznek.

KlÃ³nozÃ¡s lehetsÃ©ges az â=â segÃtsÃ©gÃ©vel, viszont ilyenkor az objektumok ugyan lemÃ¡solÃ³dnak, de a referenciÃ¡juk ugyanarra a memÃ³riaterÃ¼letre fog mutatni, azaz, ha pl. az egyik mÃ¡solt objektum egyik adattagjÃ¡t mÃ³dosÃtjuk, az az eredeti objektumra is hatÃ¡ssal lesz.

Java: ValÃ³di mÃ¡solÃ¡st Java-ban a clone() metÃ³dussal tudunk vÃ©grehajtani. Az osztÃ¡lynak, amit szeretnÃ©nk klÃ³nozhatÃ³vÃ¡ tenni implementÃ¡lnia kell a Cloneable interfÃ©szt Ã©s meg kell hÃvnia az Ås clone() metÃ³dusÃ¡t (super.clone()).

C++: C++-ban a valÃ³s klÃ³nozÃ¡s megvalÃ³sÃtÃ¡sÃ¡ra szolgÃ¡t a copy konstruktor. A copy konstruktor paramÃ©tereinek szÃ¡ma 1, ennek az egy paramÃ©ternek a tÃpusa pedig a tartalmazÃ³ osztÃ¡lyra mutatÃ³ referencia tÃpus.

Dinamikus, lokÃ¡lis Ã©s statikus objektumok lÃ©trehozÃ¡sa:

C++:

A statikusan lÃ©trehozott objektum az adott kÃ³d blokk vÃ©gÃ©n megszÅ±nik, amelyikben lÃ©tre lett hozva.

LokÃ¡lis objektumokat default paramÃ©ter vagy objektumokat tartalmazÃ³ kifejezÃ©sekben hasznÃ¡lhatunk. SzokÃ¡s mÃ©g objektum konstansnak is nevezni Åket.

Objektumokat dinamikusan a new operÃ¡tor segÃtsÃ©gÃ©vel tudunk lÃ©trehozni, amelynek tÃ¶rlÃ©sÃ©rÅl a programozÃ³nak kell gondoskodnia.

Java: Java-ban minden objektum dinamikusan jÃ¶n lÃ©tre a heap-ben.

A statikus adattagok Ã©s metÃ³dusok

A statikus adattagok Ã©s metÃ³dusok hasonlÃ³an mÅ±kÃ¶dnek Java-ban Ã©s C++-ban. MindkÃ©t nyelven a static mÃ³dosÃtÃ³val tudjuk jelezni, hogy az adott member statikus lesz.

Az ilyen adattagok Ã©s metÃ³dusok csak egy pÃ©ldÃ¡nyban jÃ¶nnek lÃ©tre Ã©s az osztÃ¡lynak lesznek a tagjai, amelyet az objektumok kÃ¶zÃ¶sen hasznÃ¡lhatnak.

A statikus metÃ³dusok nem lehetnek virtuÃ¡lisak, nem hivatkozhatnak az adott objektumra (this-re).

Az ilyen adattagok, metÃ³dusok pÃ©ldÃ¡nyosÃtÃ¡s nÃ©lkÃ¼l is hasznÃ¡lhatÃ³ak.

Olyan esetekben lehetnek hasznosak, amikor az adott adattag, metÃ³dus fÃ¼ggetlen az objektumoktÃ³l, Ã©s mindenhol megegyezne az implementÃ¡ciÃ³. PÃ©ldÃ¡ul van egy statikus adattagunk, amely a diÃ¡kok szÃ¡mÃ¡t tÃ¡rolja Ã©s egy statikus metÃ³dus, amely visszaadja ennek a statikus adattagnak az Ã©rtÃ©kÃ©t.

â

OperÃ¡ciÃ³ Ã©s operÃ¡tor overloading

OperÃ¡ciÃ³ kiterjesztÃ©s

Az operÃ¡ciÃ³ kiterjesztÃ©s mind Java, mind C++ nyelven tÃ¡mogatott Ã©s hasonlÃ³ mÃ³don mÅ±kÃ¶dik. A lÃ©nyege, hogy azonos nevÅ± fÃ¼ggvÃ©nyek tÃ¶bbszÃ¶r vannak implementÃ¡lva melyek paramÃ©terei eltÃ©rÅ szÃ¡mÃºak Ã©s tÃpusÃºak lehetnek. Ilyenek a vÃ¡ltozÃ³ paramÃ©terÅ± konstruktorok is tÃ¶bbek kÃ¶zÃ¶tt.

A fordÃtÃ³ a kiterjesztett metÃ³dusokat a paramÃ©terlistÃ¡juk alapjÃ¡n kÃ¼lÃ¶nbÃ¶zteti meg

```
void sum(int a,int b){ System.out.println(a+b); } void sum(int a,int b,int c){ System.out.println(a+b+c); }
```

OperÃ¡tor kiterjesztÃ©s

Java-ban nincs lehetÅsÃ©g az operÃ¡torok kiterjesztÃ©sÃ©re. A C++ programozÃ¡si nyelv lehetÅsÃ©get biztosÃt arra, hogy az osztÃ¡lyokra kiterjesszÃ¼k a nyelvben definiÃ¡lt binÃ¡ris Ã©s unÃ¡ris operÃ¡torokat. A kiterjesztÃ©sre vonatkozÃ³an tÃ¶bb megszorÃtÃ¡s is van, ennek ellenÃ©re ez a szolgÃ¡ltatÃ¡s jelentÅs lÃ©pÃ©s az absztrakciÃ³ nÃ¶velÃ©sÃ©nek irÃ¡nyÃ¡ba.

- A kiterjesztÃ©s CSAK osztÃ¡lyok esetÃ©n lehetsÃ©ges (ebben benne van a class, struct Ã©s a union), viszont nem mÅ±kÃ¶dik tÃ¶mbÃ¶kre, pointerekre.
- Bizonyos elemi operÃ¡torok kiterjesztÃ©sÃ©re nincs lehetÅsÃ©g, ilyenek a . (member selection), :: (scope resolution), ? : (ternary), .* (pointer to member), # Ã©s ## a preprocesszorbÃ³l. KiterjeszthetÅ viszont a (typecast) operÃ¡tor!
- Az operÃ¡torok precedenciÃ¡ja nem vÃ¡ltoztathatÃ³ meg.
- Az operÃ¡tor eljÃ¡rÃ¡sok Ã¶rÃ¶klÅdnek (kivÃ©ve az â=â operÃ¡tor).
- Az operÃ¡torok egyik operandusa osztÃ¡ly (vagy osztÃ¡lyra mutatÃ³ referencia tÃpus) kell legyen. EttÅl fÃ¼ggetlenÃ¼l lehet a kÃ©t operandus kÃ¼lÃ¶nbÃ¶zÅ.

A friend osztÃ¡lyok Ã©s eljÃ¡rÃ¡sok

Az operÃ¡tor eljÃ¡rÃ¡sok implementÃ¡ciÃ³jakor szÃ¼ksÃ©g lehet objektumok private Ã©s protected adattagjainak elÃ©rÃ©sÃ©re. Az adattagok elÃ©rÃ©sÃ©re hasznÃ¡lhatunk segÃ©d eljÃ¡rÃ¡sokat, azonban itt egy speciÃ¡lis esetrÅl van szÃ³, ahol szÃ¡mÃtÃ¡sba jÃ¶het egy âbarÃ¡tâ eljÃ¡rÃ¡s alkalmazÃ¡sa is (talÃ¡n szÃ¡ndÃ©kaink szerint metÃ³dussal szerettÃ¼k volna megvalÃ³sÃtani az operÃ¡tor eljÃ¡rÃ¡st, csak valamiÃ©rt ez nem volt megengedett).

- A friend eljÃ¡rÃ¡st az osztÃ¡ly belsejÃ©ben kell deklarÃ¡lni.
- Nemcsak eljÃ¡rÃ¡s lehet friend, hanem mÃ¡sik osztÃ¡ly is!
- A friend eljÃ¡rÃ¡sok nem lesznek az osztÃ¡ly tagjai!
- Abban az osztÃ¡lyban kell Åket deklarÃ¡lni, amelynek a tagjait el kÃvÃ¡njÃ¡k Ã©rni.
- A friend eljÃ¡rÃ¡sokat a global scope-ban kell megvalÃ³sÃtani.

KivÃ©telkezelÃ©s

A kivÃ©tel a program futÃ¡sa sorÃ¡n elÅÃ¡llÃ³ rendellenes Ã¡llapot, amely kÃ¶zbeavatkozÃ¡s nÃ©lkÃ¼l a program futÃ¡sÃ¡nak abnormÃ¡lis megszakadÃ¡sÃ¡t eredmÃ©nyezheti. A kivÃ©teles helyzetek kezelÃ©sÃ©nek elmulasztÃ¡sa pÃ©ldÃ¡ul a hÃ¡lÃ³zati a kommunikÃ¡ciÃ³, vagy az adatbÃ¡zis tranzakciÃ³ fÃ©lbeszakadÃ¡sÃ¡val jÃ¡rhat Ãºgy, hogy mindekÃ¶zben meghatÃ¡rozatlan Ã¡llapotba kerÃ¼l a rendszer. A lÃ©nyeg, hogy amikor egy hiba megjelenik a programban, azaz egy kivÃ©teles esemÃ©ny tÃ¶rtÃ©nik, a program normÃ¡lis vÃ©grehajtÃ¡sa megÃ¡ll, Ã©s Ã¡tadÃ³dik a vezÃ©rlÃ©s a kivÃ©telkezelÅ mechanizmusnak.

A Java kivÃ©telkezelÃ©se a C++ kivÃ©telkezelÃ©sÃ©re alapul. A kivÃ©telkezelÃ©s eszkÃ¶ze a try Ã©s a catch utasÃtÃ¡s, mÃg a manuÃ¡lis kivÃ©tel kivÃ¡ltÃ¡sra szolgÃ¡l a throw utasÃtÃ¡s. A kivÃ©telkezelÅ blokk vÃ©gÃ©n a finally mindenkÃ©ppen lefut.

A program azon rÃ©szeit, ahol a kivÃ©telek keletkezhetnek, Ã©s amiket utÃ¡na kivÃ©tel kezelÅ rÃ©szek kÃ¶vetnek, a program vÃ©dett rÃ©giÃ³inak nevezzÃ¼k. A kivÃ©tel bekÃ¶vetkezÃ©sekor a throwbeli kifejezÃ©s tÃpusÃ¡nak megfelelÅ catch blokk hÃvÃ³dik meg, ezt a veremben visszafelÃ© haladva keresi meg a rendszer. A verem tartalmÃ¡t az adott pontig kiÃ¼rÃti a rendszer, vÃ©grehajtja a catch blokkot, majd a try utÃ¡ni sorral folytatja a vÃ©grehajtÃ¡st. â

KivÃ©tel lÃ©trehozÃ¡sa

BeÃ©pÃtett kivÃ©tel osztÃ¡lyok mellett lÃ©trehozhatunk sajÃ¡tokat is. Java: Ha akarunk, akÃ¡r sajÃ¡t kivÃ©teleket is hozhatunk lÃ©tre bÃ¡rmely kivÃ©tel osztÃ¡ly specializÃ¡lÃ¡sÃ¡val. Ekkor egy osztÃ¡lyt az Exception osztÃ¡lybÃ³l kell szÃ¡rmaztatni Ã©s meg kell hÃvni az ÅsosztÃ¡ly konstrukturÃ¡t.

C++: `class MyException : public std::exception { std::string _msg; public: MyException(const std::string& msg) : _msg(msg){} virtual const char* what() const noexcept override { return _msg.c_str(); } };`# 9. Java Ã©s C++ programok fordÃtÃ¡sa Ã©s futtatÃ¡sa. Parancssori paramÃ©terek, fordÃtÃ¡si opciÃ³k, nagyobb projektek fordÃtÃ¡sa. Absztrakt-, interfÃ©sz- Ã©s generikus osztÃ¡lyok, virtuÃ¡lis eljÃ¡rÃ¡sok. A virtuÃ¡lis eljÃ¡rÃ¡sok megvalÃ³sÃtÃ¡sa, szerepe, hasznÃ¡lata

C++ fordÃtÃ¡s, futtatÃ¡s

ElÅfordÃtÃ¡s

ElsÅ lÃ©pÃ©sben az elÅfordÃtÃ³(preprocessor) a tÃ©nyleges fordÃtÃ³program futÃ¡sa elÅtt szÃ¶vegesen Ã¡talakÃtja a forrÃ¡skÃ³dot. Az elÅfordÃtÃ³ kÃ¼lÃ¶nbÃ¶zÅ szÃ¶veges vÃ¡ltoztatÃ¡sokat hajt vÃ©gre a forrÃ¡skÃ³don, elÅkÃ©szÃti azt a tÃ©nyleges fordÃtÃ¡sra. Feladatai: - Header fÃ¡jlok beszÃºrÃ¡sa. - A forrÃ¡sfÃ¡jlban fizikailag tÃ¶bb sorban elhelyezkedÅ forrÃ¡skÃ³d logikailag egy sorbatÃ¶rtÃ©nÅ csoportosÃtÃ¡sa (ha szÃ¼ksÃ©ges). - A kommentek helyettesÃtÃ©se whitespace karakterekkel. - Az elÅfordÃtÃ³nak a programozÃ³ Ã¡ltal megadott feladatok vÃ©grehajtÃ¡sa (szimbÃ³lumokbehelyettesÃtÃ©se, feltÃ©teles fordÃtÃ¡s, makrÃ³k, stb.) A leggyakoribb mÅ±veletei a szÃ¶veghelyettesÃtÃ©s (#define), a szÃ¶veges Ã¡llomÃ¡ny beÃ©pÃtÃ©se (#include) valamint a program rÃ©szeinek feltÃ©teltÅl fÃ¼ggÅ megtartÃ¡sa - Az elÅfeldolgozÃ³ az #include direktÃva hatÃ¡sÃ¡ra az utasÃtÃ¡sban szereplÅ szÃ¶veges fÃ¡jl tartalmÃ¡t beszÃºrja a programunkba, a direktÃva helyÃ©re.

FordÃtÃ¡s

FordÃtÃ¡skor a forrÃ¡sfÃ¡jlokbÃ³l az elsÅ lÃ©pÃ©sben tÃ¡rgymodulok (.o) keletkeznek, Ã¶nmagukban nem futÃ³kÃ©pesek. Ezt kÃ¶vetÅen szÃ¼ksÃ©g van egy szerkesztÅre, ami ezeket a modulokat Ã¶sszeszerkeszti. Linux/Unix rendszerek esetÃ©n a fordÃtÃ³ a gcc. Az alÃ¡bbi mÃ³don tudjuk lefordÃtani a tÃ¶bb forrÃ¡sfÃ¡jlbÃ³l Ã¡llÃ³ projektet: gcc -o prog main.cpp class1.cpp class2.cpp Felsoroljuk azokat a fÃ¡jlokat (a felsorolÃ¡s sorrendje lÃ©nyegtelen), amiket le szeretnÃ©nk fordÃtani. Fontos a main.cpp megadÃ¡sa hiszen ez a program belÃ©pÃ©si pontja. A -o prog, megadÃ¡sakor megadhatjuk a program nevÃ©t, ekkor prog nÃ©ven hozza lÃ©tre az .exe fÃ¡jlt. Ha nem mondunk semmit, akkor az alapÃ©rtelmezett exe fÃ¡jl neve a.out lesz. CÃ©lszerÅ± hasznÃ¡lni a -o kapcsolÃ³t. Az exe kiterjesztÃ©s csak Windows esetÃ©n van, Linux esetÃ©n csak futtatÃ¡si jogÃº fÃ¡jlt kapunk. A fordÃtÃ³ elÅszÃ¶r mindegyiket lefordÃtja, melyek a .o kiterjesztÃ©sÅ± tÃ¡rgymodul fÃ¡jlok lesznek, majd ezek Ã¶sszeszerkesztÃ©sre kerÃ¼lnek

FordÃtÃ¡si lehetÅsÃ©gek

- forrÃ¡sfÃ¡jlokbÃ³l kiindulva: gcc -o prog class1.cpp class2.cpp
    - Ekkor modulonkÃ©nt lÃ©trejÃ¶nnek a tÃ¡rgymodulok .o kiterjesztÃ©ssel.
    - Amennyiben tÃ¶bb forrÃ¡sfÃ¡jl van, akkor megoldhatÃ³: gcc -o prog *.cpp -kÃ©nt is.
- tÃ¡rgymodul Ã©s forrÃ¡sfÃ¡jl megadÃ¡sÃ¡val: Amely modulok nem vÃ¡ltoztak meg, azokat felesleges ÃºjrafordÃtani, tehÃ¡t megadhatjuk tÃ¡rgymodul Ã©s forrÃ¡sfÃ¡jl megadÃ¡sÃ¡val
    - F: gcc -o prog class1.o class2.cpp
- tÃ¡rgymodulkÃ¶nyvtÃ¡r Ã©s forrÃ¡sfÃ¡jl felhasznÃ¡lÃ¡sÃ¡val:
    - a tÃ¡rgymodulkÃ¶nyvtÃ¡r kiterjesztÃ©se .a
    - TÃ¡rgymodulkÃ¶nyvtÃ¡rat lÃ©trehozni (archiver) (-cr : create): ar -cr liba.a a.o
    - F: gcc -o prog b.cpp liba.a
- csak tÃ¡rgymodulok felhasznÃ¡lÃ¡sÃ¡val: ekkor a -c kapcsolÃ³val csak fordÃtÃ¡st vÃ©gzÃ¼nk, szerkesztÃ©st nem.
    - F: gcc -c a.cpp b.cpp: Ekkor a.o Ã©s b.o tÃ¡rgymodulokat kapunk
    - Ezt kÃ¶vetÅen az ld nevÅ± (link editor) szerkesztÅprogrammal kell Ã¶sszeszerkeszteni a modulokat.
    - F: ld -o prog a.o b.o

A gcc fordÃtÃ³ fontosabb fordÃtÃ¡si opciÃ³i

Szintaxis: gcc [kapcsolÃ³k] forrÃ¡sfÃ¡jlok - -Ob[szint]: A gcc fordÃtÃ³nak a -Ob[szint] kapcsolÃ³val tudjuk megmondani, hogy milyen optimalizÃ¡lÃ¡sokat alkalmazzon, a szint maximum 3 lehet (0,1,2), inline eljÃ¡rÃ¡sok. - -c: mint compile, lefordÃtja Ã©s Ã¶sszeÃ¡llÃtja a forrÃ¡st, linkelÃ©st nem vÃ©gez. - -o: lehetÅsÃ©gÃ¼nk van megadni a futtathatÃ³ Ã¡llomÃ¡ny nevÃ©t, amennyiben nem adunk meg, az alapÃ©rtelmezett az a.out lesz. - -Wall: A figyelmeztetÃ©seket Ãrja ki. - -g: engedÃ©lyezi a hibakeresÃ©si informÃ¡ciÃ³k elhelyezÃ©sÃ©t a programban, ami emiatt sokkal nagyobb lesz, de nyomon lehet kÃ¶vetni a futÃ¡sÃ¡t pÃ©ldÃ¡ul a gdb programmal.

C++ parancssori paramÃ©terek

int main(int argc, char* argv[])

A C++ programok kezdÅ eljÃ¡rÃ¡sa minden esetben a main() eljÃ¡rÃ¡s. A main fÃ¼ggvÃ©ny elsÅ kÃ©t paramÃ©tere az argc, ami egy int Ã©s az argv tÃ¶mb: - az argc a parancssorban szereplÅ argumentumok szÃ¡ma, - az argv a string alakban tÃ¡rolt argumentumok cÃmeit tÃ¡rolÃ³ tÃ¶mb, az elsÅ argumentum cÃme argv[0], a mÃ¡sodikÃ© argv[1], â¦, az utolsÃ³ argumentum utÃ¡n egy NULL pointer kÃ¶vetkezik. Az argv[0] a program nevÃ©t Ã©s Ãºtvonalat tartalmazza. A paramÃ©terek valÃ³jÃ¡ban az 1 indextÅl kezdÅdnek.

Java fordÃtÃ¡s, futtatÃ¡s:

Ahhoz, hogy Java programokat tudjunk futtatni, illetve fejleszteni, szÃ¼ksÃ©gÃ¼nk lesz egy fordÃtÃ³- Ã©s/vagy futtatÃ³kÃ¶rnyezetre, valamint egy fordÃtÃ³programra. A kÃ©sz programunk futtatÃ¡sÃ¡hoz mindÃ¶sszesen a JRE (Java Runtime Environment) szÃ¼ksÃ©ges, ami biztosÃtja a Java alkalmazÃ¡sok futtatÃ¡sÃ¡nak minimÃ¡lis kÃ¶vetelmÃ©nyeit, mint pÃ©ldÃ¡ul a JVM (Java Virtual Machine) Azonban a fejlesztÃ©shez szÃ¼ksÃ©gÃ¼nk lesz a JDK-ra (Java Development Kit) is. Ez tartalmazza a Java alkalmazÃ¡sok futtatÃ¡sÃ¡hoz, valamint azok kÃ©szÃtÃ©sÃ©hez, fordÃtÃ¡sÃ¡hoz szÃ¼ksÃ©ges programozÃ³i eszkÃ¶zÃ¶ket is (tehÃ¡t a JRE-t nem kell kÃ¼lÃ¶n letÃ¶lteni, a JDK tartalmazza). A fordÃtÃ¡s folyamata az alÃ¡bbiak alapjÃ¡n tÃ¶rtÃ©nik: - ElÅszÃ¶r a .java kiterjesztÃ©sÅ± fÃ¡jlokat a Java-fordÃtÃ³ (compiler) egy kÃ¶zbÃ¼lsÅ nyelvre fordÃtja - Java bÃ¡jtkÃ³dot kapunk eredmÃ©nyÃ¼l (ez a bÃ¡jtkÃ³d hordozhatÃ³). A java bÃ¡jtkÃ³d a szÃ¡mÃtÃ³gÃ©p szÃ¡mÃ¡ra mÃ©g nem Ã©rtelmezhetÅ. (kiterjesztÃ©se .class) - Ennek a kÃ³dnak az Ã©rtelmezÃ©sÃ©t Ã©s fordÃtÃ¡sÃ¡t gÃ©pi kÃ³dra a JVM (Java Virtual Machine) vÃ©gzi el futÃ¡sidÅben.

FordÃtÃ¡s: javac filename.java FuttatÃ¡s: java filename Java fordÃtÃ¡si opciÃ³k: - -g: debug informÃ¡ciÃ³k generÃ¡lÃ¡sa - -s : a generÃ¡lt fÃ¡jlok kÃ¶nyvtÃ¡rÃ¡nak megadÃ¡sa - -sourcepath : a forrÃ¡sfÃ¡jlok elÃ©rÃ©si utvonalÃ¡t meg lehet adni - -Werror: figyelmeztetÃ©s esetÃ©n megÃ¡ll a fordÃtÃ¡s Java parancssori paramÃ©terek public static void main(String[] args) A main fÃ¼ggvÃ©ny paramÃ©tere az args string tÃ¶mb, amely tartalmazza a parancssori paramÃ©tereket. Ezen a tÃ¶mbÃ¶n valamilyen ciklus segÃtsÃ©gÃ©vel vÃ©gig iterÃ¡lhatunk Ã©s a parancsori paramÃ©tereket tetszÃ©s szerint kezelhetjÃ¼k.

Nagyobb projektek esetÃ©n szokÃ¡s build fÃ¡jlokat alkalmazni: ant, gradle, makefile, stb.

VirtuÃ¡lis eljÃ¡rÃ¡sok

Egy virtuÃ¡lis eljÃ¡rÃ¡s cÃmÃ©nek meghatÃ¡rozÃ¡sa indirekt mÃ³don, futÃ¡s kÃ¶zben tÃ¶rtÃ©nik. Java-ban eleve csak virtuÃ¡lis eljÃ¡rÃ¡sok vannak (kivÃ©ve a final metÃ³dusokat, amelyeket nem lehet felÃ¼ldefiniÃ¡lni Ã©s a private metÃ³dusokat, amelyeket nem lehet Ã¶rÃ¶kÃ¶lni)

C++-ban a virtuÃ¡lis fÃ¼ggvÃ©nytÃ¡bla tartja nyilvÃ¡n a virtuÃ¡lis eljÃ¡rÃ¡sok cÃmeit. A VFT tÃ¡blÃ¡zat Ã¶rÃ¶klÅdik, feltÃ¶ltÃ©sÃ©rÅl a konstruktor gondoskodik. A szÃ¡rmaztatott osztÃ¡ly konstruktora mÃ³dosÃtja a virtuÃ¡lis fÃ¼ggvÃ©nytÃ¡blÃ¡t, kijavÃtja az ÅsosztÃ¡lybÃ³l Ã¶rÃ¶kÃ¶lt metÃ³duscÃmeket. Amikor a konstruÃ¡lÃ¡si folyamat vÃ©get Ã©r, a VFT tÃ¡blÃ¡zat minden sora Ã©rtÃ©ket kap, mÃ©gpedig a tÃ©nylegesen lÃ©trehozott osztÃ¡lynak megfelelÅ metÃ³dus cÃmeket. A VFT tÃ¡blÃ¡zat sorai ezutÃ¡n mÃ¡r nem vÃ¡ltoznak meg.

- VirtuÃ¡lis eljÃ¡rÃ¡sokat a virtual kulcsszÃ³val tudunk lÃ©trehozni. Az ÃºjrafelhasznÃ¡lÃ¡s sorÃ¡n nagy valÃ³szÃnÅ±sÃ©ggel mÃ³dosÃtÃ¡sra kerÃ¼lÅ eljÃ¡rÃ¡sokat a szÃ¼lÅ osztÃ¡lyokban cÃ©lszerÅ± egybÅl virtuÃ¡lisra megÃrni, mert ezzel jelentÅs munkÃ¡t lehet megtakarÃtani a kÃ©sÅbbiekben. â Java: Absztrakt osztÃ¡lyok
- Az abstract kulcsszÃ³val hozhatÃ³ lÃ©tre.
- Egy absztrakt osztÃ¡lybÃ³l nem hozhatÃ³ lÃ©tre objektum.
- Tartalmazhat absztrakt metÃ³dusokat (absztrakt metÃ³dusnak nincs implementÃ¡ciÃ³ja, azaz tÃ¶rzse), illetve nem absztraktokat
- Gyerek osztÃ¡lyban az abstract metÃ³dusokat felÃ¼l KELL definiÃ¡lni, ha pÃ©ldÃ¡nyosÃthatÃ³ osztÃ¡lyt szeretnÃ©nk
- Ha egy osztÃ¡ly rendelkezik legalÃ¡bb egy absztrakt metÃ³dussal, akkor osztÃ¡lynak is absztraktnak kell lennie
- Lehetnek adattagjai

InterfÃ©sz - Az interface kulcsszÃ³val lehet lÃ©trehozni - Egy speciÃ¡lis absztrakt osztÃ¡ly - Nincsenek sem megvalÃ³sÃtott metÃ³dusok, sem adattagok. CsupÃ¡n metÃ³dus deklarÃ¡ciÃ³kat tartalmaz - GyerekosztÃ¡lyban az implements kulcsszÃ³val lehet implementÃ¡lni

C++: Absztrakt osztÃ¡lyok: A tÃ¶rzs nÃ©lkÃ¼li virtuÃ¡lis eljÃ¡rÃ¡sokat pure virtual eljÃ¡rÃ¡soknak nevezzÃ¼k (pl.: virtual int getArea() = 0;). A pure virtual eljÃ¡rÃ¡s egy Ã¼res (NULL) bejegyzÃ©st foglal el a VFT (Virtual Function Table) tÃ¡blÃ¡zatban. Ha egy osztÃ¡ly ilyen eljÃ¡rÃ¡st tartalmaz, akkor azt absztrakt osztÃ¡lynak nevezzÃ¼k amiatt, mert ebbÅl az osztÃ¡lybÃ³l objektum pÃ©ldÃ¡nyokat lÃ©trehozni nem lehet. A gyermek osztÃ¡lyokban minden pure virtual eljÃ¡rÃ¡st megfelelÅ tÃ¶rzzsel kell ellÃ¡tni, ezt a fordÃtÃ³ ellenÅrzi. AmÃg egyetlen pure virtual eljÃ¡rÃ¡s is marad, az osztÃ¡ly absztrakt lesz.

â

Generikus osztÃ¡lyok

Az generikus programozÃ¡s mÃ³dszere a kÃ³d hatÃ©konysÃ¡gÃ¡nak nÃ¶velÃ©se Ã©rdekÃ©ben valÃ³sul meg. Az Ã¡ltalÃ¡nos programozÃ¡s lehetÅvÃ© teszi a programozÃ³ szÃ¡mÃ¡ra, hogy Ã¡ltalÃ¡nos algoritmust Ãrjon, amely minden adattÃpussal mÅ±kÃ¶dik. Nincs szÃ¼ksÃ©g tÃ¶bb, kÃ¼lÃ¶nfÃ©le algoritmusok lÃ©trehozÃ¡sÃ¡ra, ha az adattÃpus egÃ©sz szÃ¡m, karakterlÃ¡nc vagy karakter.

Java LehetÅsÃ©g nyÃlt az osztÃ¡lyok paramÃ©terezÃ©sÃ©re mÃ¡s tÃpusokkal. Gyakorlatilag statikus polimorfizmusrÃ³l van szÃ³, egy tÃpusparamÃ©tert adunk meg, mivel az osztÃ¡ly maga Ãºgy lett megÃrva, hogy a lehetÅ legÃ¡ltalÃ¡nosabb legyen, Ã©s ne kelljen kÃ¼lÃ¶n IntegerList, StringList, AllatList, stb. osztÃ¡lyokat megÃrnunk, hanem egy Ã¡ltalÃ¡nos osztÃ¡lyt, mint sablont hasznÃ¡lunk, Ã©s a tÃ©nyleges tÃpust a kacsacsÅrÃ¶k kÃ¶zÃ¶tt mondjuk meg. PrimitÃv tÃpussal nem lehet paramÃ©terezni, az fordÃtÃ¡si hibÃ¡t okoz.

A tÃpusparamÃ©tereket konvenciÃ³ szerint egyetlen nagybetÅ±vel szokÃ¡s elnevezni, hogy egyÃ©rtelmÅ±en megkÃ¼lÃ¶nbÃ¶ztethetÅ legyen. Gyakori elnevezÃ©sek: - E : Element (tÃ¡rolÃ³k hasznÃ¡latÃ¡nÃ¡l) - K : Key - N : Number - T : Type - V : Value

NÃ©ha szÃ¼ksÃ©g lehet, hogy a tÃpusparamÃ©terre valamilyen megszorÃtÃ¡st tegyÃ¼nk:

- public class NaturalNumber
- Wildcard-ok, ismeretlen tÃpusok:
  - public void process(List<? extends Foo> list)
    - minden olyan listÃ¡ra, ami vagy a Foo, vagy annak leszÃ¡rmazottaibÃ³l Ã¡ll
  - public void addNumbers(List<? super Foo> list)
    - minden olyan listÃ¡ra, ami vagy a Foo, vagy annak ÅseibÅl Ã¡ll

C++ C++-ban generikus osztÃ¡lyokat sablonok (template) segÃtsÃ©gÃ©vel tudunk lÃ©trehozni. A fÃ¼ggvÃ©nysablonok speciÃ¡lis funkciÃ³k, amelyek genrikus tÃpusokkal mÅ±kÃ¶dhetnek. Ez lehetÅvÃ© teszi szÃ¡munkra, hogy lÃ©trehozzunk egy fÃ¼ggvÃ©nysablont, amelynek funkcionalitÃ¡sa egynÃ©l tÃ¶bb tÃpushoz vagy osztÃ¡lyhoz igazÃthatÃ³ anÃ©lkÃ¼l, hogy megismÃ©telnÃ©nk az egyes tÃpusok teljes kÃ³djÃ¡t.

10. A programozÃ¡si nyelvek csoportosÃtÃ¡sa (paradigmÃ¡k), az egyes csoportokba tartozÃ³ nyelvek legfontosabb tulajdonsÃ¡gai

ParadigmÃ¡k

A programozÃ¡si paradigma egy osztÃ¡lyozÃ¡si forma, amely a programozÃ¡si nyelvek jellemzÅin alapul. - ImperatÃv, amelyben a programozÃ³ utasÃtja a gÃ©pet az Ã¡llapotÃ¡nak megvÃ¡ltoztatÃ¡sÃ¡ra - ProcedurÃ¡lis, amely az utasÃtÃ¡sokat eljÃ¡rÃ¡sokba csoportosÃtja - ObjektumorientÃ¡lt, amely az utasÃtÃ¡sokat csoportosÃtja az alap azon rÃ©szÃ©vel egyÃ¼tt, amelyen mÅ±kÃ¶dnek - Smalltalk - PÃ¡rhuzamos - Occam - DekleratÃv, amelyben a programozÃ³ deklarÃ¡lja a kÃvÃ¡nt eredmÃ©ny tulajdonsÃ¡gait, de nem azt, hogy hogyan kell azt kiszÃ¡mÃtani - FunkcionÃ¡lis, amelynÃ©l a kÃvÃ¡nt eredmÃ©nyt fÃ¼ggvÃ©nyalkalmazÃ¡skÃ©nt deklarÃ¡lnak - Haskell - Logikai, amelyben a kÃvÃ¡nt eredmÃ©nyt a tÃ©nyek Ã©s szabÃ¡lyok rendszerÃ©vel kapcsolatos kÃ©rdÃ©sre adott vÃ¡laszkÃ©nt deklarÃ¡lnak - Prolog - Matematikai, amelyben a kÃvÃ¡nt eredmÃ©nyt egy optimalizÃ¡lÃ¡si problÃ©ma megoldÃ¡sakÃ©nt deklarÃ¡lnak

ObjektumorientÃ¡lt paradigma

Az objektum orientÃ¡l paradigma az objektumok fogalmÃ¡n alapulÃ³ programozÃ¡si paradigma. Az objektumok egysÃ©gbe foglaljÃ¡k az adatokat Ã©s a hozzÃ¡juk tartozÃ³ mÅ±veleteket. A program egymÃ¡ssal kommunikÃ¡lÃ³ objektumok Ã¶sszessÃ©gÃ©bÅl Ã¡ll melyek hasznÃ¡ljÃ¡k egymÃ¡s mÅ±veleteit Ã©s adatait.

Smalltalk

GNU Smalltalk interpreter Beolvas minden karaktert az elsoÌ ! âig. A â!â jellel jelezzuÌk, hogy veÌgre szeretneÌnk hajtani az addig beiÌrt kifejezeÌseket. ToÌbb kifejezeÌs futtataÌsa eseteÌn itt is â mint sok maÌs nyelven â jeleznuÌnk kell azt, hogy hol fejezoÌdik be egy kifejezeÌs erre valoÌ a âpontâ (.)

Precedencia

Ha nem zÃ¡rÃ³jelezÃ¼nk â mindig balrÃ³l jobbra tÃ¶rtÃ©nik, Ãgy a 2+3*10 Ã©rtÃ©ke 50 lesz, hasznÃ¡ljunk zÃ¡rÃ³jelet: 2+(3*10). Objektumok, Ã¼zenetek A Smalltalk nyelv egy objektumorientÃ¡lt nyelv ï MINDENT objektumnak tekintÃ¼nk. A programozÃ¡s sorÃ¡n Ã¼zeneteket kÃ¼ldÃ¼nk az egyes objektumoknak. Egy objektumnak hÃ¡romfÃ©le Ã¼zenetet kÃ¼ldhetÃ¼nk: - UnÃ¡ris: szintaxis: âHelloâ printNl ! - BinÃ¡ris: szintaxis: 3+5 - Kulcsszavas: szintaxis: tomb at:1 put: 10 Objektumok Ã¶sszehasonlÃtÃ¡sa: kÃ©t objektum egyenlÅ, ha ugyanazt az objektumot reprezentÃ¡ljÃ¡k Ã©s azonos, ha Ã©rtÃ©kÃ¼k megegyezik Ã©s egyazon objektumok.

Objektumok mÃ¡solÃ¡sa

- deepCopy (unÃ¡ris Ã¼zenet): Teljes mÃ¡solat kÃ©szÃtÃ©se objektumrÃ³l.
- shallowCopy (unÃ¡ris Ã¼zenet): FelszÃni mÃ¡solat
- copy (unÃ¡ris Ã¼zenet): OsztÃ¡lyonkÃ©nt vÃ¡ltozÃ³ lehet, az Object osztÃ¡lyban a shallowCopy-t jelenti.

MetaosztÃ¡ly

Mint korÃ¡ban emlÃtettÃ¼k, a Smalltalkban mindent objektumnak tekintÃ¼nk. MÃ©g az osztÃ¡lyok is objektumok. De ha az osztÃ¡ly objektum, akkor az is - mint minden mÃ¡s objektum - valamilyen osztÃ¡lyhoz kell tartozzon. MÃ¡skÃ©pp fogalmazva minden osztÃ¡ly (pontosan) egy mÃ¡sik osztÃ¡ly pÃ©ldÃ¡nya. Ezen "mÃ¡sik" osztÃ¡lyt metaosztÃ¡lynak hÃvjuk

Object osztÃ¡ly

Az Object osztÃ¡ly minden osztÃ¡ly kÃ¶zÃ¶s Åse, tehÃ¡t minden objektum az Object osztÃ¡ly egy pÃ©ldÃ¡nya. EzÃ©rt minden, az Object osztÃ¡lynak rendelkezÃ©sre Ã¡llÃ³ mÅ±velettel minden mÃ¡s objektum is rendelkezik. - class â unÃ¡ris: visszatÃ©rÃ©se az objektum osztÃ¡lya - isMemberOf â kulcsszavas: visszatÃ©rÃ©se logikai Ã©rtÃ©k. Ha a cÃmzett objektum pÃ©ldÃ¡nya ezen osztÃ¡lynak, akkor "true" a visszatÃ©rÃ©si Ã©rtÃ©k, egyÃ©bkÃ©nt "false" - 'Hello' isMemberOf: String ! â true

VÃ¡ltozÃ³k

- LokÃ¡lis vÃ¡ltozÃ³k:
  - |x y z| - deklarÃ¡lÃ¡sa (2 pipeline kÃ¶zÃ¶tt)
  - x := 2. (egyszeres Ã©rtÃ©kadÃ¡s)
  - x := y := z := 2. (tÃ¶bbszÃ¶rÃ¶s Ã©rtÃ©kadÃ¡s)
- GlobÃ¡lis vÃ¡ltozÃ³k: Smalltalk at: #valtozonev put: Ã©rtÃ©k !

Blokkok

MÃ¡s programozÃ¡si nyelveken megismert programblokkok szerepÃ©vel egyezik meg. Vannak paramÃ©teres Ã©s nem paramÃ©teres blokkok. ParamÃ©teres blokkok rendelkeznek lokÃ¡lis vÃ¡ltozÃ³kkal, melyeknek a blokk kiÃ©rtÃ©kelÃ©sekor adunk Ã©rtÃ©ket. A vÃ¡ltozÃ³k Ã©lettartama Ã©s lÃ¡thatÃ³sÃ¡ga korlÃ¡tozÃ³dik az adott blokkra. - [:i | i printNl] value: 5 - [âHelloâ print . âworldâ printNl] value.

VezÃ©rlÃ©si szerkezetek

- FeltÃ©teles vezÃ©rlÃ©s: valtozo > 10 ifTrue: [âx erteke nagyobb 10-nelâ printNl] ifFalse: [âx erteke nem nagyobb 10-nelâ printNl]
- IsmÃ©tlÃ©ses vezÃ©rlÃ©s: [a<10] whileTrue: [a printNl . a:=a+1]
- For ciklus: 1 to: 10 do: [:i | i printNl] KollekciÃ³k
- Set: ismÃ©tlÃ©s nÃ©lkÃ¼li rendezetlen halmaz - new, add()
    -
- Bag: olyan Set, amiben megengedjÃ¼k az ismÃ©tlÅdÃ©st - new, add()
    -
- Dictionary: egy asszociatÃv tÃ¶mb (egy olyan tÃ¶mb, amit nem csak szÃ¡mokkal, hanem (itt) tetszÅleges objektummal is indexelhetÃ¼nk)
    - TÃ¶mb
- tÃ¶mb := Array new: 10
- tÃ¶mb at: 1
- tÃ¶mb at: 1 put: obj A collect
- kollekciÃ³ elemein lÃ©pked vÃ©gig, mely minden egyes elemÃ©re vÃ©grehajtja az Ã¼zenet argumentumblokkjÃ¡ban talÃ¡lhatÃ³ utasÃtÃ¡sokat
- |tomb| tomb := #(10 3 43 29) collect: [:tombelem | tombelem*2] OsztÃ¡lyok
- pÃ©ldÃ¡nyvÃ¡ltozÃ³k: minden objektum rendelkezik vele
- osztÃ¡lyvÃ¡ltozÃ³: kb. statikus globÃ¡lis vÃ¡ltozÃ³

MetÃ³dusok definiÃ¡lÃ¡sa osztÃ¡lyokhoz

pl.:

BeolvasÃ¡s x := stdin nextLine.S Integer Ã¼zenetek

FunkcionÃ¡lis programozÃ¡s

- A funkcionÃ¡lis programnyelvek a programozÃ¡si feladatot egy fÃ¼ggvÃ©ny kiÃ©rtÃ©kelÃ©sÃ©nek tekintik.
- minden fÃ¼ggvÃ©ny
- A kÃ©t fÅ eleme az Ã©rtÃ©k Ã©s a fÃ¼ggvÃ©ny, nevÃ©t is fÃ¼ggvÃ©nyek kitÃ¼ntetett szerepÃ©nek kÃ¶szÃ¶nheti.
- Egy mÃ¡s megfogalmazÃ¡s szerint, a funkcionÃ¡lis programozÃ¡s sorÃ¡n a programozÃ³ inkÃ¡bb azt specifikÃ¡lja programban, mit kell kiszÃ¡mÃtani, nem azt, hogy hogyan, milyen lÃ©pÃ©sekben.
- FÃ¼ggvÃ©nyek hÃvÃ¡sÃ¡bÃ³l Ã©s kiÃ©rtÃ©kelÃ©sbÅl Ã¡ll a program. Nincsenek Ã¡llapotok, mellÃ©khatÃ¡sok (nem szÃ¡mÃt, mikor, csak az melyik fÃ¼ggvÃ©nyt hÃvjuk).

Haskell

Egy funkcionÃ¡lis programozÃ¡si nyelven Ãrt programban nem a kifejezÃ©sek egymÃ¡sutÃ¡njÃ¡n van a hangsÃºly. A program egy fÃ¼ggvÃ©nyhÃvÃ¡ssal hajtÃ³dik vÃ©gre. Egy funkcionÃ¡lis program tÃpus- , osztÃ¡ly-, Ã©s fÃ¼ggvÃ©nydeklarÃ¡ciÃ³k, illetve definÃciÃ³k sorozatÃ¡bÃ³l Ã©s egy kezdeti kifejezÃ©s kiÃ©rtÃ©kelÃ©sÃ©bÅl Ã¡ll. A kiÃ©rtÃ©kelÃ©st Ãºgy kÃ©pzeljÃ¼k el, mint a kezdeti kifejezÃ©sben szereplÅ fÃ¼ggvÃ©nyek behelyettesÃtÃ©sÃ©t. TehÃ¡t egy program vÃ©grehajtÃ¡sa nem mÃ¡s, mint a kezdeti kifejezÃ©sbÅl kiindulÃ³ redukciÃ³s lÃ©pÃ©sek sorozata. Egy kifejezÃ©s normÃ¡l formÃ¡jÃº, ha mÃ¡r tovÃ¡bb nem redukÃ¡lhatÃ³ (nem Ã¡tÃrhatÃ³) Ã¡llapotban van. Egy redukÃ¡lhatÃ³ kifejezÃ©st redexnek hÃvunk. KiÃ©rtÃ©kelÃ©si mÃ³dok

A Haskell nyelv a lusta kiÃ©rtÃ©kelÃ©si stratÃ©giÃ¡t hasznÃ¡lja. A lusta kiÃ©rtÃ©kelÃ©s sorÃ¡n mindig a legkÃ¼lsÅ redex kerÃ¼l helyettesÃtÃ©sre, az argumentumokat csak szÃ¼ksÃ©g esetÃ©n Ã©rtÃ©keli ki. Ez a mÃ³dszer mindig megtalÃ¡lja a kezdeti kifejezÃ©s normÃ¡l formÃ¡jÃ¡t. A mohÃ³ kiÃ©rtÃ©kelÃ©s az argumentumok kiÃ©rtÃ©kelÃ©sÃ©vel kezdÅdik, csak ezutÃ¡n hajtja vÃ©gre a fÃ¼ggvÃ©ny alkalmazÃ¡sÃ¡nak megfelelÅ redukciÃ³s lÃ©pÃ©st. FuttatÃ¡s ElindÃtjuk a Haskell interpretert (hugs) Ã©s betÃ¶ltjÃ¼k az Ã¡ltalunk megÃrt definÃciÃ³s forrÃ¡sÃ¡llomÃ¡nyt. BetÃ¶ltÃ©s utÃ¡n rendelkezÃ©sre Ã¡ll az Ã¶sszes Ã¡ltalunk megÃrt fÃ¼ggvÃ©ny, melyek kÃ¶zÃ¼l bÃ¡rmelyiket meghÃvhatjuk a fÃ¼ggvÃ©ny nevÃ©nek beÃrÃ¡sÃ¡val (a megfelelÅ paramÃ©terezÃ©ssel). Amennyiben mÃ³dosÃtjuk a definÃciÃ³s Ã¡llomÃ¡nyt, Ãºjra kell tÃ¶lteni azt.

Atomi tÃpusok: Int, Float, Bool FÃ¼ggvÃ©nyek definiÃ¡lÃ¡sa

A visszatÃ©rÃ©si Ã©rtÃ©ket a kiÃ©rtÃ©kelÃ©se hatÃ¡rozza meg, ami lehet egy konstans Ã©rtÃ©k vagy akÃ¡r egy rekurzÃv kifejezÃ©s is

EsetvizsgÃ¡latok

FÃ¼ggvÃ©ny paramÃ©tere fÃ¼ggvÃ©ny

LokÃ¡lis definÃciÃ³k fÃ¼ggvÃ©nydefinÃciÃ³kban

TÃpusok lÃ©trehozÃ¡sa

PÃ©ldÃ¡k

Logikai programozÃ¡s

A problÃ©makÃ¶rrel kapcsolatos tÃ©nyeket logikai kÃ©pletek formÃ¡jÃ¡ban fejezik ki, Ã©s a programokat kÃ¶vetkeztetÃ©si szabÃ¡lyok alkalmazÃ¡sÃ¡val hajtjÃ¡k vÃ©gre, amÃg nem talÃ¡lnak vÃ¡laszt a problÃ©mÃ¡ra, vagy a kÃ©pletek halmaza nem kÃ¶vetkezetes.

Prolog

A logikai programok egy modellre vonatkoztatott Ã¡llÃtÃ¡sok halmaza, melyek a modell tulajdonsÃ¡gait Ã©s azok kÃ¶zÃ¶tt fellÃ©pÅ kapcsolatokat (relÃ¡ciÃ³it) ÃrjÃ¡k le. Egy adott relÃ¡ciÃ³t meghatÃ¡rozÃ³ Ã¡llÃtÃ¡sok rÃ©szhalmazÃ¡t predikÃ¡tumnak nevezzÃ¼k. A predikÃ¡tumokat alkotÃ³ Ã¡llÃtÃ¡sok tÃ©nyek vagy szabÃ¡lyok lehetnek. A tÃ©nyeket Ã©s szabÃ¡lyokat (Ã©s majd a Prolognak feltett kÃ©rdÃ©seket is) ponttal zÃ¡rjuk le. TekintsÃ¼k a kÃ¶vetkezÅ pÃ©ldÃ¡t, mely egy csalÃ¡d tagjai kÃ¶zÃ¶tt fellÃ©pÅ kapcsolatot ÃrjÃ¡k le.

A szulo predikÃ¡tum argumentumait szÃ¡ndÃ©kosan Ãrtuk kis betÅ±kkel. A kis betÅ±kkel Ãrtakat a Prolog konstanskÃ©nt kezeli. (ka, katalin, szilvia, stbâ¦) Minden nyomtatott nagybetÅ±t vagy nagy kezdÅbetÅ±vel kezdÅdÅket vÃ¡ltozÃ³nak tekinti. (X, Y, Szilvia, Magdolna, stbâ¦)

FuttatÃ¡s

- kiterjesztÃ©s .pl
- A Prolog egy terminÃ¡lablakba beÃrt âsicstusâ paranccsal indÃthatÃ³. Egy Prolog Ã¡llomÃ¡nyt a kÃ¶vetkezÅkÃ©ppen âtÃ¶lthetjÃ¼k beâ: (feltÃ©ve, hogy az aktuÃ¡lis kÃ¶nyvtÃ¡rban lÃ©tezik egy prolog.pl Ã¡llomÃ¡ny)

A Prolog program felÃ©pÃtÃ©se

Termek - EgyszerÅ± termek -
- Ãsszetett termek - Lista: nagyon hasonlÃt a Haskell-ben megismert listÃ¡ra. Itt sincsenek indexelve az elemek, rekurziÃ³val fogjuk bejÃ¡rni a listÃ¡t. PÃ©lda listÃ¡ra: [1,2,3,4,5]. KiÃ©rtÃ©kelÃ©s KifejezÃ©sek kiÃ©rtÃ©kelÃ©sÃ©re a beÃ©pÃtett, infix is operÃ¡tort hasznÃ¡lhatjuk. ÃltalÃ¡nos alakja:

PÃ©ldÃ¡k

PÃ¡rhuzamos programozÃ¡s

Occam

Az Occam egy pÃ¡rhuzamos programozÃ¡si nyelv. Ezen paradigma szerint az egyes folyamatok pÃ¡rhuzamosan futnak. Ez tÃ¶bb processzoros gÃ©pek esetÃ©n valÃ³s pÃ¡rhuzamossÃ¡got jelent (egy processzor egy folyamatot dolgoz fel), de egy processzor esetÃ©n ez nyilvÃ¡n nem valÃ³sulhat meg, az egyes folyamatok âidÅszeleteketâ kapnak, az Occam a pÃ¡rhuzamossÃ¡got idÅosztÃ¡ssal szimulÃ¡lja. Az egyes folyamatok kÃ¶zÃ¶tti kommunikÃ¡ciÃ³ csatornÃ¡kon keresztÃ¼l valÃ³sul meg. A P1 Ã©s P2 folyamatok a C csatornÃ¡n keresztÃ¼l kommunikÃ¡lnak:

A folyamatok kÃ¶zÃ¶tti kommunikÃ¡ciÃ³t mindig csatornÃ¡kkal valÃ³sÃtjuk meg. A fenti pÃ©ldÃ¡ban a P1 folyamat a C csatornÃ¡n keresztÃ¼l valamilyen adatot kÃ¼ld a P2 folyamatnak. Ez a kÃ¶vetkezÅkÃ©ppen valÃ³sul meg: ha egy folyamat elÃ©rkezik arra a pontra, ahol Ã©rtÃ©ket kÃ¼ld [fogad], vÃ¡rakozik a mÃ¡sik folyamatra, amÃg az is el nem Ã©r a fogad [kÃ¼ld] pontra. Amikor mindketten kÃ©szen Ã¡llnak az adatcserÃ©re (azaz mindkÃ©t folyamatban a kÃ¼ldÃ©s [fogadÃ¡s] pontra kerÃ¼lt a vezÃ©rlÃ©s) lÃ©trejÃ¶n az adatcsere, majd mindkettÅ folytatja a futÃ¡sÃ¡t.

FordÃtÃ¡s

- KroC, csak Linux-hoz
- kroc -d pelda.occ Fontos tudnivalÃ³k a nyelvrÅl
- Minden, a nyelvben lefoglalt kulcsszoÌt nagy betuÌvel kell iÌrni (SEQ, PAR, PROC, stb...)
- A blokkstruktuÌraÌt indentaÌcioÌval jeloÌljuÌk (keÌt szoÌkoÌzzel beljebb kezdjuÌk)
- Minden egyes kifejezeÌs uÌj sorban kezdoÌdik (esetlegesen keÌt szoÌkoÌzzel beljebb)
- Egy Occam program a koÌvetkezoÌkeÌppen eÌpuÌl fel:
- PeÌldaÌul:

Elemi folyamatok

A fenti pÃ©ldÃ¡ban, kÃ¼ldÃ©s esetÃ©ben egy kifejezÃ©st (k + 5) kÃ¼ldÃ¼nk a C csatornÃ¡ra, fogadÃ¡s esetÃ©n pedig a C csatornÃ¡rÃ³l vÃ¡runk egy Ã©rtÃ©ket, amely az x vÃ¡ltozÃ³ban kerÃ¼l. A SKIP folyamat a legegyszerÅ±bb elemi folyamat, âsemmit nem csinÃ¡lâ. Haszontalannak tÅ±nhet, de Ã¶sszetettebb programok esetÃ©ben (pÃ©ldÃ¡ul mÃ©g nem kifejlesztett programrÃ©szek esetÃ©ben) hasznos lehet. PÃ¡rhuzamos folyamatok esetÃ©ben fontos, hogy minden folyamat terminÃ¡ljon, ellenkezÅ esetben az egÃ©sz, folyamatokbÃ³l Ã¡llÃ³ ârendszerâ leÃ¡ll. A STOP szintÃ©n ânem csinÃ¡l semmitâ, de ez sosem terminÃ¡l â ellentÃ©tben a SKIP-el. Egy folyamatban a STOP (feltÃ©ve hogy a vezÃ©rlÃ©s odakerÃ¼l), annak holtpontba jutÃ¡sÃ¡t eredmÃ©nyezi. SzintÃ©n haszontalannak tÅ±nhet, de ezzel egy folyamatot leÃ¡llÃthatunk mÃ¡s folyamatok mÅ±kÃ¶dÃ©sÃ©nek befolyÃ¡solÃ¡sa nÃ©lkÃ¼l, ami hibakeresÃ©snÃ©l hasznos lehet. Azt mondjuk, hogy egy folyamat holtpont Ã¡llapotba kerÃ¼lt, ha az mÃ¡r nem kÃ©pes tovÃ¡bbi mÅ±kÃ¶dÃ©sre (vezÃ©rlÃ©se leÃ¡ll), Ã©s ez a leÃ¡llÃ¡s nem a folyamat helyes lefutÃ¡sÃ¡nak eredmÃ©nye. PÃ¡rhuzamos folyamatok kÃ¶zÃ¼l akÃ¡r egy folyamat holtpont Ã¡llapotba kerÃ¼lÃ©se az egÃ©sz program holtpont Ã¡llapotba kerÃ¼lÃ©sÃ©t eredmÃ©nyezi, hiszen az Ã¶sszes tÃ¶bbi folyamat vÃ¡rja a holtpontban levÅ folyamat terminÃ¡lÃ¡sÃ¡t, ami sosem fog bekÃ¶vetkezni. Blokk struktÃºra 2 szÃ³kÃ¶zÃ¶nkÃ©nt beljebb kell kezdeni

Precedencia

A kifejezÃ©sekben, operÃ¡torok kÃ¶zÃ¶tt precedenciÃ¡t nem hatÃ¡rozunk meg, Ãgy MINDIG zÃ¡rÃ³jelezÃ©st kell hasznÃ¡lni a precedencia meghatÃ¡rozÃ¡sÃ¡hoz

AdattÃpusok

Csatorna
SEQ
PAR

Az egÃ©sz PAR blokk akkor terminÃ¡l, ha a benne âelindÃtottâ folyamatok mindegyike terminÃ¡l PROC A PROC egy elÅre definiÃ¡lt, nÃ©vvel ellÃ¡tott folyamat. TekinthetÃ¼nk Ãºgy rÃ¡, mintha egy eljÃ¡rÃ¡st definiÃ¡lnÃ¡nk

ALT

Ha egy År engedÃ©lyezettÃ© vÃ¡lik, akkor a benne megadott vÃ¡ltozÃ³ felveszi a csatornÃ¡rÃ³l Ã©rkezÅ adat Ã©rtÃ©kÃ©t Ã©s âelindÃtjaâ a hozzÃ¡ tartozÃ³ folyamatot Az x vÃ¡ltozÃ³ Ã©rtÃ©ke attÃ³l fÃ¼gg, hogy c1-re vagy c2-re Ã©rkezik elÅbb adat. Mivel a program ÃrÃ¡sakor nem tudhatjuk, hogy melyik csatornÃ¡rÃ³l fog adat Ã©rkezni, ezÃ©rt az ALT-ot tartalmazÃ³ programok nemdeterminisztikusak FÃ¼ggvÃ©ny

VezÃ©rlÃ©si szerkezetek - FeltÃ©teles vezÃ©rlÃ©s Holtpont elkerÃ¼lÃ©se
- IsmÃ©tlÃ©ses vezÃ©rlÃ©s

- For ciklus

PÃLDÃK HIÃNYOZNAK, KELL EGYÃLTALÃN?# 11. SzoftverfejlesztÃ©si folyamat Ã©s elemei; a folyamat kÃ¼lÃ¶nbÃ¶zÅ modelljei

AlapvetÅ elemek

- SzoftverspecifikÃ¡ciÃ³: a szoftver funkciÃ³it Ã©s korlÃ¡tait meg kell hatÃ¡rozni
- SzoftvertesztelÃ©s Ã©s implementÃ¡ciÃ³: a specifikÃ¡ciÃ³nak megfelelÅen a szoftvert elÅ kell Ã¡llÃtani
- SzoftvervalidÃ¡ciÃ³: a szoftvert ellenÅrizni kell, hogy tÃ©nyleg azt fejlesztettÃ¼k ki, amit az Ã¼gyfÃ©l kÃvÃ¡n
- SzoftverevolÃºciÃ³: a szoftvert Ãºgy alakÃtani, hogy megfeleljen a kÃ©sÅbbi kÃvÃ¡nsÃ¡goknak

A szoftverfolyamat modelljei

A szoftverfolyamat modellje a szoftverfolyamat absztrakt reprezentÃ¡ciÃ³ja. Ezek a modellek egy-egy egyedi perspektÃvÃ¡bÃ³l reprezentÃ¡l egy szoftverfolyamatot, de nem pontos specifikÃ¡ciÃ³ja annak. Sokkal inkÃ¡bb hasznos absztrakciÃ³k, amit a szoftverfejlesztÃ©si folyamat kÃ¼lÃ¶nbÃ¶zÅ megkÃ¶zelÃtÃ©si mÃ³djainak megÃ©rtÃ©sÃ©hez hasznÃ¡lunk.

VÃzesÃ©s modell

- SpecifikÃ¡ciÃ³: rÃ¶gzÃtjÃ¼k a termÃ©k kÃ¶vetelmÃ©nyeit. Mit tudjon a szoftver, Ã©s mit nem.
- TervezÃ©s: szÃ©tvÃ¡lasztÃ³dnak a szoftver- Ã©s hardverkÃ¶vetelmÃ©nyek. MegtervezzÃ¼k a rendszer architektÃºrÃ¡jÃ¡t.
- ImplementÃ¡ciÃ³: a szoftver fejlesztÃ©se, egysÃ©gtesztelÃ©se. Az egysÃ©gtesztelÃ©s azt a cÃ©lt szolgÃ¡lja, hogy a szoftver minden egyes egysÃ©ge megfelel-e a specifikÃ¡ciÃ³nak.
- VerifikÃ¡ciÃ³: a kÃ¼lÃ¶nÃ¡llÃ³ programegysÃ©ges Ã©s programok integrÃ¡lÃ¡sa Ã©s teljes rendszerkÃ©nt valÃ³ tesztelÃ©se, hogy a rendszer megfelel-e a specifikÃ¡ciÃ³nak. A tesztelÃ©s utÃ¡n a rendszer Ã¡tadhatÃ³ az Ã¼gyfÃ©lnek.
- KarbantartÃ¡s: a szoftver Ã©letciklusÃ¡nak leghosszabb fÃ¡zisa. A karbantartÃ¡sba beletartozik olyan hibÃ¡k javÃtÃ¡sa is, amelyek nem merÃ¼ltek fel az Ã©letciklus korÃ¡bbi szakaszaiban, illetve a szolgÃ¡ltatÃ¡sok tovÃ¡bbfejlesztÃ©se.

A fÃ¡zisok eredmÃ©nye egy vagy tÃ¶bb dokumentum, amelyek jÃ³vÃ¡hagyÃ¡sa megtÃ¶rtÃ©nt. A kÃ¶vetkezÅ fÃ¡zis nem indulhat, amÃg az elÅzÅ be nem fejezÅdÃ¶tt.

ProblÃ©ma: a folyamat korai szakaszaiban Ã¡llÃ¡st kell foglalnunk Ã©s el kell kÃ¶teleznÃ¼nk magunkat, Ã©s nehÃ©z az Ã¼gyfÃ©lhez tÃ¶rtÃ©nÅ alkalmazkodÃ¡s. Akkor jÃ³, ha elÅre ismerjÃ¼k a kÃ¶vetelmÃ©nyeket. Nagyobb rendszerek kisebb folyamatainÃ¡l hasznÃ¡ljÃ¡k fÅleg.

EvolÃºciÃ³s fejlesztÃ©s

Az evolÃºciÃ³s fejlesztÃ©s lÃ©nyege, hogy ki kell fejleszteni egy korai implementÃ¡ciÃ³t, azt a felhasznÃ¡lÃ³kkal vÃ©lemÃ©nyeztetni, Ã©s finomÃtani a felhasznÃ¡lÃ³i visszajelzÃ©sek alapjÃ¡n, amÃg megfelelÅ rendszert el nem Ã©rÃ¼nk.

KÃ©t kÃ¼lÃ¶nbÃ¶zÅ tÃpusa ismert:

- FeltÃ¡rÃ³ fejlesztÃ©s: a folyamat cÃ©lja az hogy a megrendelÅvel egyÃ¼tt feltÃ¡rjuk a kÃ¶vetelmÃ©nyeket, Ã©s kialakÃtsuk a vÃ©glekges rendszert. A vÃ©gleges rendszer Ãºgy alakul ki, hogy egyre tÃ¶bb, az Ã¼gyfÃ©l Ã¡ltal kÃ©rt tulajdonsÃ¡got tÃ¡rsÃtunk a mÃ¡r meglÃ©vÅkhÃ¶z.
- EldobhatÃ³ prototÃpus fejlesztÃ©se: ekkor az evolÃºciÃ³s fejlesztÃ©s cÃ©lja, hogy minÃ©l jobban megÃ©rtsÃ¼k az Ã¼gyfÃ©l kÃ¶vetelmÃ©nyeit, Ã©s azokra alapozva a legpontosabban fejlesszÃ¼k le a termÃ©ket.

Az evolÃºciÃ³s fejlesztÃ©s jobb, mint a vÃzesÃ©s modell, ha a lehetÅ legpontosabban szeretnÃ©nk az Ã¼gyfÃ©l kÃvÃ¡nsÃ¡gainak megfelelÅ szoftvert fejleszteni. ElÅnye, hogy a specifikÃ¡ciÃ³ inkrementÃ¡lisan fejleszthetÅ.

A vezetÅsÃ©g Ã©s a tervezÅk szempontjÃ¡bÃ³l kÃ©t problÃ©ma merÃ¼lhet fel:

- A folyamat nem lÃ¡thatÃ³. A menedzsereknek rendszeresen leszÃ¡llÃthatÃ³ eredmÃ©nyekre van szÃ¼ksÃ©gÃ¼k, hogy mÃ©rhessÃ©k a fejlÅdÃ©st.
- A rendszerek sokszor szegÃ©nyesen struktÃºrÃ¡ltak. A folyamatos vÃ¡ltoztatÃ¡sok rontjÃ¡k a szoftver struktÃºrÃ¡jÃ¡t.

A vÃ¡rhatÃ³an rÃ¶vid Ã©lettartamÃº kis vagy kÃ¶zepes rendszerek esetÃ©n az evolÃºciÃ³s megkÃ¶zelÃtÃ©si mÃ³d a legcÃ©lravezetÅbb.

IterÃ¡ciÃ³s, inkrementÃ¡lis modell

- Folyamat iterÃ¡ciÃ³ja elkerÃ¼lhetetlen
- ha a kÃ¶vetelmÃ©nyek vÃ¡ltoznak, akkor a folyamat bizonyos rÃ©szeit is vÃ¡ltoztatni kell
- ennÃ©l a modellnÃ©l minimÃ¡lis a specifikÃ¡ciÃ³, fejlesztÃ©sben sok iterÃ¡ciÃ³ van, Ã©s menet kÃ¶zben alakul ki a vÃ©gleges specifikÃ¡ciÃ³
- InkrementalitÃ¡s: rÃ©szfunkciÃ³kkal mÃ¡r mÅ±kÃ¶dÅ rendszert fejlesztÃ¼nk, amit minden iterÃ¡ciÃ³ban (inkrementÃ¡lisan) javÃtunk
- Nagy kÃ¶rvonalakban specifikÃ¡ljuk a rendszert
    - âInkremensekâ meghatÃ¡rozÃ¡sa
    - FunkcionalitÃ¡sokhoz prioritÃ¡sokat rendelÃ¼nk
    - Magasabbakat elÅbb kell biztosÃtani
- ArchitektÃºrÃ¡t meg kell hatÃ¡rozni
- TovÃ¡bbi inkremensek pontos specifikÃ¡lÃ¡sa menet kÃ¶zben tÃ¶rtÃ©nik
- Egyes inkremensek kifejlesztÃ©se tÃ¶rtÃ©nhet akÃ¡r kÃ¼lÃ¶nbÃ¶zÅ folyamatokkal is - VÃzesÃ©s vagy evolÃºciÃ³s, amelyik jobb
- Az elkÃ©szÃ¼lt inkremenseket akÃ¡r szolgÃ¡latba is lehet Ã¡llÃtani
- Ha hatÃ¡ridÅ csÃºszÃ¡s van kilÃ¡tÃ¡sban a teljes projekt nem lesz kudarcra Ãtélve, esetleg csak egyes inkremensek
- MegfelelÅ mÃ©retÅ± inkremensek meghatÃ¡rozÃ¡sa nem triviÃ¡lis feladat
    - Ha tÃºl kicsi: nem mÅ±kÃ¶dÅkÃ©pes
    - Ha tÃºl nagy: elveszÃtjÃ¼k a modell lÃ©nyegÃ©t

Bizonyos esetekben szÃ¡mos alapvetÅ funkcionalitÃ¡st kell megvalÃ³sÃtani. EgÃ©sz addig nincs mÅ±kÃ¶dÅ inkremens

eXtreme Programming (XP)

- SzÃ©lsÅsÃ©ges inkrementÃ¡lis modell
- Nagyon kis funkcionalitÃ¡sÃº inkremensek
- MegrendelÅ intenzÃv rÃ©szvÃ©tele fontos
- ProgramozÃ¡s csoportos tevÃ©kenysÃ©g - tÃ¶bben Ã¼lnek a kÃ©pernyÅ elÅtt
- Sok tÃ¡madÃ³ja van

RAD (Rapid Application Development)

- ExtrÃ©m rÃ¶vid Ã©letciklus
- MÅ±kÃ¶dÅ rendszer 60-90 nap alatt
- VÃzesÃ©s modell ânagysebessÃ©gÅ±â adaptÃ¡lÃ¡sa
- PÃ¡rhuzamos fejlesztÃ©s
- Komponens alapÃº fejlesztÃ©s
- FÃ¡zisai:
    - Ãzleti modellezÃ©s
        - Milyen informÃ¡ciÃ³k Ã¡ramlanak funkciÃ³k kÃ¶zÃ¶tt
    - AdatmodellezÃ©s
        - FinomÃtÃ¡s adatszerkezetekre

    - Adatfolyam processzus
        - Adatmodell megvalÃ³sÃtÃ¡sa
    - AlkalmazÃ¡s generÃ¡lÃ¡s
        - 4GT alkalmazÃ¡sa, automatikus generÃ¡lÃ¡s, komponensek
    - TesztelÃ©s
        - Csak komponens tesztelÃ©s
- Nagy emberi erÅforrÃ¡sigÃ©ny
- FejlesztÅk Ã©s megrendelÅk intenzÃv egyÃ¼ttmÅ±kÃ¶dÃ©se szÃ¼ksÃ©ges
- Nem minden tÃpusÃº fejlesztÃ©snÃ©l alkalmazhatÃ³

SpirÃ¡lis modell

- Olyan evolÃºciÃ³s modell, amely kombinÃ¡lja a prototÃpus modellt a vÃzesÃ©s modellel
- InkrementÃ¡lis modellhez hasonlÃ³, csak Ã¡ltalÃ¡nosabb megfogalmazÃ¡sban
- Nincsenek rÃ¶gzÃtett fÃ¡zisok
- MÃ¡s modelleket Ã¶lelhet fel
    - PrototÃpuskÃ©szÃtÃ©s pontatlan kÃ¶vetelmÃ©nyek esetÃ©n
    - VÃzesÃ©s modell egy kÃ©sÅbbi kÃ¶rben
    - Kritikus rÃ©szek esetÃ©n formÃ¡lis mÃ³dszerek
- A spirÃ¡l kÃ¶rei a folyamat egy-egy fÃ¡zisÃ¡t reprezentÃ¡ljÃ¡k
- Minden kÃ¶rben a kimenet egy âreleaseâ (modell vagy szoftver)
- KÃ¶rÃ¶k cÃ©ljai pl.:
    - MegvalÃ³sÃthatÃ³sÃ¡g (elvi prototÃpusok)
    - KÃ¶vetelmÃ©nyek meghatÃ¡rozÃ¡sa (prototÃpusok)
    - TervezÃ©s (modellek Ã©s inkremensek)
    - Stb. (javÃtÃ¡s, karbantartÃ¡s, stb.)
- A kÃ¶rÃ¶k 3-6 darab szektorokra oszthatÃ³k

WINWIN spirÃ¡lis modell

- WINWIN = mindenki nyer
- MegrendelÅ Ã©s fejlesztÅ is
- Sok tÃ¡rgyalÃ¡s kell a kÃ©t fÃ©l kÃ¶zÃ¶tt
- WINWIN modell szÃ¡mos tÃ¡rgyalÃ¡si szempontot visz bele a spirÃ¡lis modellbe
    - Egyes (al)rendszerek kulcsszereplÅi, Ã©rdekeltek
    - Az Ã©rdekeltek nyerÅ feltÃ©telei
    - TÃ¡rgyalÃ¡s, kompromisszumok

12. Projektmenedzsment. KÃ¶ltsÃ©gbecslÃ©s, szoftvermÃ©rÃ©s

Projektmenedzsment

ÃsszetevÅi:

- Az emberek menedzselÃ©se
- MinÅsÃ©g-ellenÅrzÃ©s Ã©s -biztosÃtÃ¡s
- Folyamat tovÃ¡bbfejlesztÃ©se
- KonfigurÃ¡ciÃ³ kezelÃ©s
- Rendszer Ã©pÃtÃ©s
- Hibamenedzsment

Projekt sikertelensÃ©gÃ©nek okai

- A szÃ¼ksÃ©ges rÃ¡fordÃtÃ¡sok alulbecslÃ©se
- Technikai nehÃ©zsÃ©gek
- A projekt csapatban nem megfelelÅ a kommunikÃ¡ciÃ³
- A projekt menedzsment hibÃ¡i

Az Emberek menedzselÃ©se

SzoftverfejlesztÅ szervezet legnagyobb vagyona az emberek Sok projekt bukÃ¡sÃ¡nak legfÅbb oka a rossz humÃ¡nmenedzsment HatÃ©kony egyÃ¼ttmÅ±kÃ¶dÃ©s fontos - Csapatszellemet kell kialakÃtani Fontos a kommunikÃ¡ciÃ³ Az emberek kivÃ¡lasztÃ¡sa kÃ¼lÃ¶nbÃ¶zÅ tesztekkel tÃ¶rtÃ©nhet:

- ProgramozÃ¡si kÃ©pessÃ©g
- Pszichometrikus tesztek

MinÅsÃ©g-ellenÅrzÃ©s Ã©s âbiztosÃtÃ¡s

Mindenki cÃ©lja: termÃ©k vagy szolgÃ¡ltatÃ¡s minÅsÃ©gÃ©nek magas szinten tartÃ¡sa A termÃ©k feleljen meg a specifikÃ¡ciÃ³nak FejlesztÅnek is lehetnek (belsÅ) igÃ©nyei, pl. karbantarthatÃ³sÃ¡g Egyes jellemzÅket nem kÃ¶nnyÅ± specifikÃ¡lni , pl. karbantarthat Ã³sÃ¡g

Folyamat tovÃ¡bbfejlesztÃ©se

CMM(I) (Capability Maturity Model (Integration)): a szoftver folyamat mÃ©rÃ©se

- CÃ©l: a szoftverfejlesztÃ©si folyamat hatÃ©konysÃ¡gÃ¡nak mÃ©rÃ©se
- Egy szervezet megkaphatja valamely szintÅ± minÅsÃtÃ©sÃ©t
- 5 besorolÃ¡si szint
    -
        1. Kezdeti: csak nÃ©hÃ¡ny folyamat definiÃ¡lt, a tÃ¶bbsÃ©gÃ¼k esetleges
    -
        1. ReprodukÃ¡lhatÃ³: az alapvetÅ projekt menedzsment folyamatok definiÃ¡ltak. KÃ¶ltsÃ©g Ã¼temezÃ©s, funkcionalitÃ¡s kezelÃ©se
    -
        1. DefiniÃ¡lt: a menedzsment Ã©s a fejlesztÃ©s folyamatai is dokumentÃ¡ltak Ã©s szabvÃ¡nyosÃtottak az egÃ©sz szervezetre.
    -
        1. EllenÅrzÃ¶tt: a szoftver folyamat Ã©s termÃ©k minÅsÃ©gÃ©nek rÃ©szletes mÃ©rÃ©se, ellenÅrzÃ©se.
    -
        1. OptimalizÃ¡lt: a folyamatok folytonos javÃtÃ¡sa az Ãºj technolÃ³giÃ¡k ellenÅrzÃ¶tt bevezetÃ©sÃ©vel

A szoftver folyamat javÃtÃ¡sa - Az alapvetÅ cÃ©l a minÅsÃ©g Ã©s a hatÃ©konysÃ¡g nÃ¶velÃ©se

HasznÃ¡ljunk metrikÃ¡kat

Hiba analÃzis

- HibÃ¡k eredetÃ©nek kategorizÃ¡lÃ¡sa
- HibÃ¡k javÃtÃ¡si kÃ¶ltsÃ©gei

KonfigurÃ¡ciÃ³ kezelÃ©s

A rendszer vÃ¡ltozÃ¡sainak kezelÃ©se

VÃ¡ltozÃ¡sok felÃ¼gyelt mÃ³don tÃ¶rtÃ©njenek

FejlesztÃ©s, evolÃºciÃ³, karbantartÃ¡s miatt van rÃ¡ szÃ¼ksÃ©g

MinÅsÃ©gkezelÃ©s rÃ©sze

Szoftver vÃ¡ltozatok

- VerziÃ³k (version, revision)
- KiadÃ¡sok (release)
- Alapvonal (baseline, mainline, trunk)
- FejlesztÃ©si Ã¡gak (branch)

KonfigurÃ¡ciÃ³s adatbÃ¡zis - mindent tÃ¡rol:

- ForrÃ¡skÃ³d, (binÃ¡ris kÃ³d), dokumentumok
- ÃpÃtÃ©si folyamat, szkriptek
- Hiba adatbÃ¡zis
- VÃ¡ltoztatÃ¡sok tÃ¶rtÃ©nete
- VerziÃ³k

Rendszer Ã©pÃtÃ©s

Komponensek fordÃtÃ¡sa Ã©s szerkesztÃ©se

Komponensek (Ã©s fÃ¡jlok) kÃ¶zÃ¶tt Ã©pÃtÃ©si fÃ¼ggÅsÃ©gek vannak

Nagy rendszernÃ©l bonyolult a folyamat - Hosszadalmas is, ezÃ©rt inkrementÃ¡lisan kell vÃ©gezni

AutomatizÃ¡lni kell: Ã©pÃtÃ©si szkriptek: configure, make

EszkÃ¶zkivÃ¡lasztÃ¡s (fordÃtÃ³program), beÃ¡llÃtÃ¡sok

Hibamenedzsment

HibÃ¡k kÃ¶vetÃ©se fontos

Fontos, mert sok hiba van/lesz: kategorizÃ¡lÃ¡s, prioritÃ¡sok felÃ¡llÃtÃ¡sa, kÃ¶vetÃ©s elengedhetetlen

Milyen jellegÅ± a hiba - (hibabejelentÃ©s, Ãºj feature, ...)

HibÃ¡k kÃ¶vetÃ©sÃ©re hibaadatbÃ¡zis

- Minden hibÃ¡nak egyedi azonosÃtÃ³ja van
- BejelentÅ neve
- KijelÃ¶lt felelÅs szemÃ©ly, megfigyelÅk listÃ¡ja
- DÃ¡tum
- RÃ¶vid Ã¶sszegzÃ©s
- SÃºlyossÃ¡g: pl. triviÃ¡lis, kicsi, nagy
- Platform, operÃ¡ciÃ³s rendszer
- TermÃ©k, komponens, verziÃ³szÃ¡m
- FÃ¼ggÅsÃ©gek mÃ¡s hibÃ¡kkal
- Fontos a hiba Ã©letÃºtjÃ¡nak rÃ¶gzÃtÃ©se

SzoftverkÃ¶ltsÃ©g becslÃ©se

Projekt tevÃ©kenysÃ©geinek kapcsolÃ³dÃ¡sa a munka-, idÅ- Ã©s pÃ©nzkÃ¶ltsÃ©gekhez BecslÃ©seket lehet Ã©s kell adni

Projekt Ã¶sszkÃ¶ltsÃ©ge:

- Hardver Ã©s szoftver kÃ¶ltsÃ©g karbantartÃ¡ssal
- UtazÃ¡si Ã©s kÃ©pzÃ©si kÃ¶ltsÃ©g
- MunkakÃ¶ltsÃ©g

Ezeket meg kell becsÃ¼lni:

- Mennyi pÃ©nz?
- Mennyi rÃ¡fordÃtÃ¡s?
- Mennyi idÅ?

MunkakÃ¶ltsÃ©g:

- LegjelentÅsebb
- FejlesztÅk fizetÃ©se
- KisegÃtÅ szemÃ©lyzet fizetÃ©se
- BÃ©rleti dÃj, rezsi
- InfrastruktÃºra hasznÃ¡lat (pl. hÃ¡lÃ³zat)
- JÃ¡rulÃ©kok, adÃ³k

SzoftvermÃ©rÃ©s

SzoftvermÃ©rÃ©s: termÃ©k vagy folyamat valamely jellemzÅjÃ©t numerikusan kifejezni (metrika). Ezen Ã©rtÃ©kekbÅl kÃ¶vetkeztetÃ©sek vonhatÃ³k le a minÅsÃ©gre vonatkozÃ³an.

KÃ©t csoport:

- VezÃ©rlÃ©si metrikÃ¡k. Folyamattal kapcsolatosak, pl. egy hiba javÃtÃ¡sÃ¡hoz szÃ¼ksÃ©ges Ã¡tlagos idÅ(folyamat Ã©s projekt metrikÃ¡k)
- Prediktor metrikÃ¡k. TermÃ©kkel kapcsolatosak, pl. LOC, ciklomatikus komplexitÃ¡s, osztÃ¡ly metÃ³dusainak szÃ¡ma (termÃ©k metrikÃ¡k)
  - LOC = Lines Of Code
    - TÃ¶bb technika: Csak nem Ã¼res sorok, Csak vÃ©grehajthatÃ³ sorok
    - FÃ©lrevezetÅ lehet - Nem Ã¶sszehasonlÃthatÃ³ programozÃ¡si nyelvek (assembly, magas szintÅ± nyelv)
- MÃ©rÃ©si folyamat:
  - AlkalmazandÃ³ mÃ©rÃ©sek kivÃ¡lasztÃ¡sa

    - MÃ©rni kÃvÃ¡nt komponensek kivÃ¡lasztÃ¡sa
    - MÃ©rÃ©s (metrika szÃ¡mÃtÃ¡s)
- TermÃ©k metrikÃ¡k
    - Dinamikus
        - Szorosabb kapcsolat egyes minÅsÃ©gi jellemzÅkkel
        - (pl. teljesÃtmÃ©ny, hibÃ¡k szÃ¡ma)
    - Statikus
        - KÃ¶zvetett kapcsolat
        - SzÃ¡mtalan konkrÃ©t metrikÃ¡t ajÃ¡nlottak mÃ¡r
        - Kritikus kÃ©rdÃ©s: hogyan kÃ¶vetkeztetÃ¼nk a minÅsÃ©gi jellemzÅkre a sok szÃ¡mbÃ³l?
    - FajtÃ¡k:
        - MÃ©ret
        - KomplexitÃ¡s, csatolÃ¡s, kohÃ©ziÃ³
        - ObjektumorientÃ¡ltsÃ¡ggal kapcsolatos metrikÃ¡k
- MÃ©ret alapÃº metrikÃ¡k (folyt.)
    - SzÃ©leskÃ¶rÅ±en hasznÃ¡ljÃ¡k ezeket a metrikÃ¡kat, de nagyon sok vita van alkalmazÃ¡sokrÃ³l
    - HibÃ¡k / KLOC
    - Defekt / KLOC
    - KÃ¶ltsÃ©g / LOC
    - DokumentÃ¡ciÃ³s oldalak / KLOC
    - HibÃ¡k / emberhÃ³nap
    - LOC / emberhÃ³nap
    - KÃ¶ltsÃ©g / dokumentÃ¡ciÃ³s oldal
- FunkciÃ³ alapÃº metrikÃ¡k
    - FelhasznÃ¡lÃ³i inputok szÃ¡ma - alkalmazÃ¡shoz szÃ¼ksÃ©ges adatok
    - FelhasznÃ¡lÃ³i outputok szÃ¡mariportok, kÃ©pernyÅk,hibaÃ¼zenetek
    - FelhasznÃ¡lÃ³i kÃ©rdÃ©sek szÃ¡ma - on-line input Ã©s output
    - FÃ¡jlok szÃ¡ma- adatok logikai csoportja
- 3D mÃ©rtÃ©k
    - SzÃ¡mÃtÃ¡s: Index=I+O+Q+F+E+T+R
        - I=input
        - O=output
        - Q=lekÃ©rdezÃ©s
        - F=fÃ¡jlok
        - E=kÃ¼lsÅ interfÃ©sz
        - T=transzformÃ¡ciÃ³
        - R=Ã¡tmenetek
- MinÅsÃ©g mÃ©rÃ©se
    - IntegritÃ¡s: kÃ¼lsÅ tÃ¡madÃ¡sok elleni vÃ©delem
    - FenyegetettsÃ©g: annak valÃ³szÃnÅ±sÃ©ge, hogy egy adott tÃpusÃº tÃ¡madÃ¡s bekÃ¶vetkezik egy adott idÅszakban
    - BiztonsÃ¡g: annak valÃ³szÃnÅ±sÃ©ge, hogy egy adott tÃpusÃº tÃ¡madÃ¡st visszaver a rendszer
    - IntegritÃ¡s = Î£ [1-(fenyegetettsÃ©g x (1-biztonsÃ¡g))] (ÃsszegzÃ©s a kÃ¼lÃ¶nbÃ¶zÅ tÃ¡madÃ¡s tÃpusokra tÃ¶rtÃ©nik)
    - DRE (defect removal efficiency)
        - DRE = E/(E+D), ahol E olyan hibÃ¡k szÃ¡ma, amelyeket mÃ©g az Ã¡tadÃ¡s elÅt felfedezÃ¼nk, D pedig az Ã¡tadÃ¡s utÃ¡n a felhasznÃ¡lÃ³ Ã¡ltal Ã©szlelt hiÃ¡nyossÃ¡gok szÃ¡ma

15. Neumann-elvÅ± gÃ©p egysÃ©gei. CPU, adatÃºt, utasÃtÃ¡s-vÃ©grehajtÃ¡s, utasÃtÃ¡s- Ã©s processzorszintÅ± pÃ¡rhuzamossÃ¡g. KorszerÅ± szÃ¡mÃtÃ³gÃ©pek tervezÃ©si elvei. PÃ©ldÃ¡k RISC (UltraSPARC) Ã©s CISC (Pentium 4) architektÃºrÃ¡kra, jellemzÅik

SzÃ¡mÃtÃ³gÃ©p architektÃºra: A hardver egy Ã¡ltalÃ¡nos absztrakciÃ³ja: a hardver struktÃºrÃ¡jÃ¡t Ã©s viselkedÃ©sÃ©t jelenti mÃ¡s rendszerek egyedi, sajÃ¡tos tulajdonsÃ¡gaitÃ³l eltekintve

Neumann elvÅ± gÃ©p

- Neumann-architektÃºra mÃ¡ra a tÃ¡rolt programÃº szÃ¡mÃtÃ³gÃ©p fogalmÃ¡vÃ¡ vÃ¡lt
- SzÃ¡mÃtÃ³gÃ©p mÅ±kÃ¶dÃ©sÃ©t tÃ¡rolt program vezÃ©rli (Turing).
- A vezÃ©rlÃ©st vezÃ©rlÃ©s-folyam (control-flow) segÃtsÃ©gÃ©vel lehet leÃrni

- Az aritmetikai Ã©s logikai mÅ±veletek (programutasÃtÃ¡sok) vÃ©grehajtÃ¡sÃ¡t Ã¶nÃ¡llÃ³ rÃ©szegysÃ©g (ALU) vÃ©gzi
- 2-es (binÃ¡ris) szÃ¡mrendszer alkalmazÃ¡sa
- Ãt funkcionÃ¡lis egysÃ©g (aritmetikai egysÃ©g, kÃ¶zponti vezÃ©rlÅegysÃ©g, memÃ³riÃ¡k, bemeneti Ã©s kimeneti egysÃ©gek)

Neumann-elvÅ± gÃ©p egysÃ©gei

- kÃ¶zponti memÃ³ria: a program kÃ³djÃ¡t Ã©s adatait tÃ¡rolja szÃ¡mokkÃ©nt
- kÃ¶zponti feldolgozÃ³egysÃ©g (CPU): a kÃ¶zponti memÃ³riÃ¡ban tÃ¡rolt program utasÃtÃ¡sait beolvassa Ã©s vÃ©grehajtja
- kÃ¼lsÅ sÃn: a rÃ©szegysÃ©geket kÃ¶ti Ã¶ssze, adatokat, cÃmeket, vezÃ©rlÅjeleket tovÃ¡bbÃt
- belsÅ sÃn: CPU rÃ©szegysÃ©gei kÃ¶zÃ¶tti kommunikÃ¡ciÃ³t hozza lÃ©tre (vezÃ©rlÅegysÃ©g-ALU-regiszterek)
- beviteli/kiviteli eszkÃ¶zÃ¶k: kapcsolatot teremt a felhasznÃ¡lÃ³val, adatot tÃ¡rol a hÃ¡ttÃ©rtÃ¡ron, nyomtat, stb.
- mÅ±kÃ¶dÃ©st biztosÃtÃ³ jÃ¡rulÃ©kos eszkÃ¶zÃ¶k: pÃ©ldÃ¡ul gÃ©phÃ¡z, tÃ¡pellÃ¡tÃ¡s, hÅ±tÃ©sâ¦

CPU, adatÃºt, utasÃtÃ¡s-vÃ©grehajtÃ¡s, utasÃtÃ¡s- Ã©s processzorszintÅ± pÃ¡rhuzamossÃ¡g

CPU

A CPU feladata a kÃ¶zponti memÃ³riÃ¡ban tÃ¡rolt program utasÃtÃ¡sainak beolvasÃ¡sa Ã©s vÃ©grehajtÃ¡sa 3 fÅ egysÃ©ge:

- vezÃ©rlÅegysÃ©g (CU):
  - UtasÃtÃ¡sok beolvasÃ¡sa a memÃ³riÃ¡bÃ³l
  - az ALU Ã©s regiszterek vezÃ©rlÃ©se
- aritmetika-logikai egysÃ©g (ALU):
  - Egy tipikus Neumann-fÃ©le CPU belsÅ szerkezetÃ©nek rÃ©szÃ©ben az ALU vÃ©gzi az Ã¶sszeadÃ¡st, a kivonÃ¡st Ã©s mÃ¡s egyszerÅ± mÅ±veleteket az inputjain, Ãgy adva Ã¡t az eredmÃ©nyt az output regiszternek, azaz a kimeneten ez fog megjelenni.
  - az utasÃtÃ¡sok vÃ©grehajtÃ¡sÃ¡hoz szÃ¼ksÃ©ges aritmetikai Ã©s logikai mÅ±veleteket vÃ©gzi el
    - Aritmetikai operÃ¡torok: +, -, *, / (alapmÅ±veletek)
    - Logikai operÃ¡torok: NOT, AND, OR, NAND, NOR, XOR, NXOR (EQ)
- regiszterek:
  - kismÃ©retÅ±, gyors memÃ³riarekeszek, amelyek rÃ©szeredmÃ©nyeket Ã©s vezÃ©rlÅinformÃ¡ciÃ³kat tÃ¡rolnak
  - A regiszterek a szÃ¡mÃtÃ³gÃ©pek kÃ¶zponti feldolgozÃ³ egysÃ©geinek, illetve mikroprocesszorainak gyorsan ÃrhatÃ³-olvashatÃ³, ideiglenes tartalmÃº, Ã©s Ã¡ltalÃ¡ban egyszerre csak 1 gÃ©pi szÃ³ feldolgozÃ¡sÃ¡ra alkalmas tÃ¡rolÃ³egysÃ©gei
- adatÃºt

AdatÃºt

- Az adatÃºt az adatok Ã¡ramlÃ¡sÃ¡nak Ãºtja, alapfeladata, hogy kivÃ¡lasszon egy vagy kÃ©t regisztert, az ALU-val mÅ±veletet vÃ©geztessen el rajtuk (Ã¶sszeadÃ¡s, kivonÃ¡s...), az eredmÃ©nyt pedig valamelyik regiszterben tÃ¡rolja. Egyes gÃ©peken az adatÃºt mÅ±kÃ¶dÃ©sÃ©t mikroprogram vezÃ©rli, mÃ¡sutt a vezÃ©rlÃ©s kÃ¶zvetlenÃ¼l a hardver feladata.
- Folyamata:
  - A regiszter kÃ©szletbÅl feltÃ¶ltÅdik az ALU kÃ©t bemenÅ regisztere (A Ã©s B)
  - Az eredmÃ©ny az ALU kimenÅ regiszterÃ©be kerÃ¼l
  - Az ALU kimenÅ regiszterÃ©bÅl a kijelÃ¶lt regiszterbe kerÃ¼l az eredmÃ©ny
- KÃ©t operandusnak az ALU-n tÃ¶rtÃ©nÅ Ã¡tfutÃ¡sÃ¡bÃ³l Ã©s az eredmÃ©ny regiszterbe tÃ¡rolÃ¡sÃ¡bÃ³l Ã¡llÃ³ folyamatot adatÃºtciklusnak nevezzÃ¼k.

UtasÃtÃ¡s-vÃ©grehajtÃ¡s

A mikroprocesszor 1-1 utasÃtÃ¡sa Ãºgynevezett gÃ©pi ciklusok egymÃ¡sutÃ¡nisÃ¡gÃ¡bÃ³l Ã¡ll, vagyis 1 utasÃtÃ¡s egy vagy tÃ¶bb gÃ©pi ciklusbÃ³l tevÅdik Ã¶ssze.

A CPU minden utasÃtÃ¡st aprÃ³ lÃ©pÃ©sek sorozatakÃ©nt hajt vÃ©gre. Ezek a lÃ©pÃ©sek a kÃ¶vetkezÅk:

```
1. A soron kÃ¶vetkezÅ utasÃtÃ¡s beolvasÃ¡sa a memÃ³riÃ¡bÃ³l az utasÃtÃ¡sregiszterbe.
2. Az utasÃtÃ¡sszÃ¡mlÃ¡lÃ³ beÃ¡llÃtÃ¡sa a kÃ¶vetkezÅ utasÃtÃ¡s cÃmÃ©re.
3. A beolvasott utasÃtÃ¡s tÃpusÃ¡nak meghatÃ¡rozÃ¡sa.
4. Ha az utasÃtÃ¡s memÃ³riabeli szÃ³t hasznÃ¡l, a szÃ³ helyÃ©nek megÃ¡llapÃtÃ¡sa.
5. Ha szÃ¼ksÃ©ges, a szÃ³ beolvasÃ¡sa a CPU egy regiszterÃ©be.
6. Az utasÃtÃ¡s vÃ©grehajtÃ¡sa.
7. Vissza az 1. pontra, a kÃ¶vetkezÅ utasÃtÃ¡s vÃ©grehajtÃ¡sÃ¡nak megkezdÃ©se.
```

Ezt a lÃ©pÃ©ssorozatot gyakran nevezik betÃ¶ltÅ-dekÃ³dolÃ³-vÃ©grehajtÃ³ ciklusnak, Ã©s kÃ¶zponti szerepet tÃ¶lt be minden szÃ¡mÃtÃ³gÃ©p mÅ±kÃ¶dÃ©sÃ©ben.

Nagy problÃ©ma a szÃ¡mÃtÃ³gÃ©peknÃ©l, hogy a memÃ³ria olvasÃ¡sa lassÃº, ezÃ©rt az utasÃtÃ¡s Ã©s az adatok beolvasÃ¡sa kÃ¶zben a CPU tÃ¶bb rÃ©sze kihasznÃ¡latlan. A gyorsÃtÃ¡s egyik mÃ³dja a lapkÃ¡k gyorsÃtÃ¡sa az Ã³rajel frekvenciÃ¡jÃ¡nak nÃ¶velÃ©sÃ©vel, de ez korlÃ¡tozott. Emiatt a legtÃ¶bb tervezÅ a pÃ¡rhuzamossÃ¡g kiaknÃ¡zÃ¡sÃ¡ban lÃ¡t lehetÅsÃ©get.

A pÃ¡rhuzamossÃ¡g kÃ©t fÃ©lekÃ©ppen lehet jelen: utasÃtÃ¡sszintÅ± pÃ¡rhuzamossÃ¡g vagy processzorszintÅ± pÃ¡rhuzamossÃ¡g formÃ¡jÃ¡ban.

UtasÃtÃ¡sszintÅ± pÃ¡rhuzamossÃ¡g

Az utasÃtÃ¡sok vÃ©grehajtÃ¡sÃ¡nak gyorsÃtÃ¡sa Ã©rdekÃ©ben elÅre be lehet olvasni az utasÃtÃ¡sokat, hogy azok rendelkezÃ©sre Ã¡lljanak, amikor szÃ¼ksÃ©g van rÃ¡juk. Ezeket az utasÃtÃ¡sokat egy elÅolvasÃ¡si puffer (prefetch buffer) elnevezÃ©sÅ± regiszterkÃ©szlet tÃ¡rolja. Ilyen mÃ³don a soron kÃ¶vetkezÅ utasÃtÃ¡st Ã¡ltalÃ¡ban az elÅolvasÃ¡si pufferbÅl lehet venni ahelyett, hogy egy memÃ³riaolvasÃ¡s befejezÅdÃ©sÃ©re kellene vÃ¡rni.

CsÅvezetÃ©k:

LÃ©nyegÃ©ben az elÅolvasÃ¡s az utasÃtÃ¡s vÃ©grehajtÃ¡sÃ¡t kÃ©t rÃ©szre osztja: beolvasÃ¡s Ã©s tulajdonkÃ©ppeni vÃ©grehajtÃ¡s. A csÅvezetÃ©k ezt a stratÃ©giÃ¡t viszi sokkal tovÃ¡bb. Az utasÃtÃ¡s vÃ©grehajtÃ¡sÃ¡t kettÅ helyett tÃ¶bb rÃ©szre osztja, minden rÃ©szt kÃ¼lÃ¶n hardverelem kezel, amelyek mind egyszerre mÅ±kÃ¶dhetnek.

A csÅvezetÃ©k lehetÅvÃ© teszi, hogy kompromisszumot kÃ¶ssÃ¼nk kÃ©sleltetÃ©s (mennyi ideig tart egy utasÃtÃ¡s vÃ©grehajtÃ¡sa) Ã©s Ã¡teresztÅkÃ©pessÃ©g (hÃ¡ny MIPS a processzor sebessÃ©ge) kÃ¶zÃ¶tt.

PÃ¡rhuzamos csÅvezetÃ©k:

Az elÅolvasÃ³ egysÃ©g kÃ©t utasÃtÃ¡st olvas be egyszerre, majd ezeket az egyik, illetve a mÃ¡sik csÅvezetÃ©kre teszi. A csÅvezetÃ©keknek sajÃ¡t ALU-juk van, Ãgy pÃ¡rhuzamosan tudnak mÅ±kÃ¶dni, feltÃ©ve, hogy a kÃ©t utasÃtÃ¡s nem hasznÃ¡lja ugyanazt az erÅforrÃ¡st, Ã©s egyik sem hasznÃ¡lja fel a mÃ¡sik eredmÃ©nyÃ©t. UgyanÃºgy, mint egyetlen csÅvezetÃ©k esetÃ©n, a feltÃ©telek betartÃ¡sÃ¡t vagy a fordÃtÃ³programnak kell garantÃ¡lnia, vagy a konfliktusokat egy kiegÃ©szÃtÅ hardvernek kell a vÃ©grehajtÃ¡s sorÃ¡n felismernie Ã©s kikÃ¼szÃ¶bÃ¶lnie.

SzuperskalÃ¡ris architektÃºra:

Itt egy csÅvezetÃ©ket hasznÃ¡lnak, de tÃ¶bb funkcionÃ¡lis egysÃ©ggel. Ezek olyan processzorok, amelyek tÃ¶bb â gyakran nÃ©gy vagy hat â utasÃtÃ¡s vÃ©grehajtÃ¡sÃ¡t kezdik el egyetlen Ã³rajel alatt. TermÃ©szetesen egy szuperskalÃ¡ris CPU-nak tÃ¶bb funkcionÃ¡lis egysÃ©gÃ©nek kell lennie, amelyek kezelik mindezeket az utasÃtÃ¡sokat. Az utasÃtÃ¡sok megkezdÃ©sÃ©t sokkal nagyobb Ã¼temben vÃ©gzik, mint amilyen Ã¼temben azokat vÃ©gre lehet hajtani, Ãgy a terhelÃ©s megoszlik a funkcionÃ¡lis egysÃ©gek kÃ¶zÃ¶tt. A szuperskalÃ¡ris processzor elvÃ©ben implicit mÃ³don benne van az a feltÃ©telezÃ©s, hogy a megfelelÅ fÃ¡zis lÃ©nyegesen gyorsabban tudja elÅkÃ©szÃteni az utasÃtÃ¡sokat, mint ahogy a rÃ¡kÃ¶vetkezÅ fÃ¡zis kÃ©pes azokat vÃ©grehajtani. Ez a fÃ¡zis funkcionÃ¡lis egysÃ©geinek tÃ¶bbsÃ©ge egy Ã³rajelnÃ©l jÃ³val tÃ¶bb idÅt igÃ©nyel feladata elvÃ©gzÃ©sÃ©hez â a memÃ³riÃ¡hoz fordulÃ³k vagy a lebegÅpontos mÅ±veleteket vÃ©gzÅk biztosan. AkÃ¡r tÃ¶bb ALU-t is tartalmazhat.

ProcesszorszintÅ± pÃ¡rhuzamossÃ¡g

TÃ¶mb processzorok:

Egy tÃ¶mbprocesszor nagyszÃ¡mÃº egyforma processzorbÃ³l Ã¡ll, ugyanazon mÅ±veleteket egyszerre vÃ©gzik kÃ¼lÃ¶nbÃ¶zÅ adathalmazokon. A feladatok szabÃ¡lyossÃ¡ga Ã©s szerkezete kÃ¼lÃ¶nÃ¶sen megfelelÅvÃ© teszi ezeket pÃ¡rhuzamos feldolgozÃ¡sra. Olyan utasÃtÃ¡sokat hajthatnak vÃ©gre, mint amilyen pÃ©ldÃ¡ul kÃ©t vektor elemeinek pÃ¡ronkÃ©nti Ã¶sszeadÃ¡sa.

Multiprocesszorok:

Ezekben tÃ¶bb teljes CPU van, amelyek egy kÃ¶zÃ¶s memÃ³riÃ¡t hasznÃ¡lnak. Amikor kÃ©t vagy tÃ¶bb CPU rendelkezik azzal a kÃ©pessÃ©ggel, hogy szorosan egyÃ¼ttmÅ±kÃ¶djenek, akkor azokat szorosan kapcsoltaknak nevezik. A legegyszerÅ±bb, ha egyetlen sÃn van, amelyhez csatlakoztatjuk a memÃ³riÃ¡t Ã©s az Ã¶sszes processzort. Ha sok gyors processzor prÃ³bÃ¡lja Ã¡llandÃ³an elÃ©rni a memÃ³riÃ¡t a kÃ¶zÃ¶s sÃnen keresztÃ¼l, az konfliktusokhoz vezet. Az egyik megoldÃ¡s, hogy minden processzornak biztosÃtunk valamekkora sajÃ¡t lokÃ¡lis memÃ³riÃ¡t, amelyet a tÃ¶bbiek nem Ã©rhetnek el. Ãgy csÃ¶kken a kÃ¶zÃ¶s sÃn forgalma. JellemzÅen maximum pÃ¡r szÃ¡z CPU-t Ã©pÃtenek Ã¶ssze.

MultiszÃ¡mÃtÃ³gÃ©pek:

NehÃ©z sok processzort Ã©s memÃ³riÃ¡t Ã¶sszekÃ¶tni. EzÃ©rt gyakran sok Ã¶sszekapcsolt szÃ¡mÃtÃ³gÃ©pbÅl Ã¡llÃ³ rendszereket Ã©pÃtenek, amelyeknek csak sajÃ¡t memÃ³riÃ¡juk van. A multiszÃ¡mÃtÃ³gÃ©pek CPU-it lazÃ¡n kapcsoltaknak nevezik. A multiszÃ¡mÃtÃ³gÃ©p processzorai Ã¼zenetek kÃ¼ldÃ©sÃ©vel kommunikÃ¡lnak egymÃ¡ssal. Nagy rendszerekben nem cÃ©lszerÅ± minden szÃ¡mÃtÃ³gÃ©pet minden mÃ¡sikkal Ã¶sszekÃ¶tni, ezÃ©rt 2 Ã©s 3 dimenziÃ³s rÃ¡csot, fÃ¡kat Ã©s gyÅ±rÅ±ket hasznÃ¡lnak. Ennek kÃ¶vetkeztÃ©ben egy gÃ©p valamelyik mÃ¡sikhoz kÃ¼ldÃ¶tt Ã¼zeneteinek gyakran egy vagy tÃ¶bb kÃ¶zbensÅ gÃ©pen vagy csomÃ³ponton kell Ã¡thaladniuk ahhoz, hogy a kiindulÃ¡si helyÃ¼krÅl elÃ©rjenek a cÃ©ljukhoz. NÃ©hÃ¡ny mikroszekundumos nagysÃ¡grendÅ± Ã¼zenetkÃ¼ldÃ©si idÅk nagyobb nehÃ©zsÃ©g nÃ©lkÃ¼l elÃ©rhetÅk. 10 000 processzort tartalmazÃ³ multiszÃ¡mÃtÃ³gÃ©peket is Ã©pÃtettek mÃ¡r.

KorszerÅ± szÃ¡mÃtÃ³gÃ©pek tervezÃ©si elvei

- Minden utasÃtÃ¡st kÃ¶zvetlenÃ¼l a hardver hajtson vÃ©gre.
    - Ezek nem bonthatÃ³k fel interpretÃ¡lt mikroutasÃtÃ¡sokra. Az interpretÃ¡ciÃ³s szint kikÃ¼szÃ¶bÃ¶lÃ©sÃ©vel a legtÃ¶bb utasÃtÃ¡s gyors lesz.
- MaximalizÃ¡lni kell az utasÃtÃ¡sok kiadÃ¡sÃ¡nak Ã¼temÃ©t
    - MegprÃ³bÃ¡ljÃ¡k egy mÃ¡sodperc alatt a lehetÅ legtÃ¶bb utasÃtÃ¡s vÃ©grehajtÃ¡sÃ¡t elkezdeni, tehÃ¡t a pÃ¡rhuzamossÃ¡gra kell tÃ¶rekedni.
- Az utasÃtÃ¡sok kÃ¶nnyen dekÃ³dolhatÃ³k legyenek.
    - Az utasÃtÃ¡sok szabÃ¡lyosak, egyforma hosszÃºak legyenek, Ã©s kevÃ©s mezÅbÅl Ã¡lljanak.
- Csak a betÃ¶ltÅ Ã©s tÃ¡rolÃ³ utasÃtÃ¡sok hivatkozzanak a memÃ³riÃ¡ra
    - A memÃ³riamÅ±veletek sok idÅt vehetnek igÃ©nybe, legjobb mÃ¡s utasÃtÃ¡sokkal Ã¡tfedve vÃ©grehajtani, ha semmi mÃ¡st nem tesznek, csak adatokat mozgatnak a regiszterek Ã©s a memÃ³ria kÃ¶zÃ¶tt, minden mÃ¡s utasÃtÃ¡s csak regisztereket hasznÃ¡lhat.
- Sok regiszter legyen.
    - Mivel a memÃ³riamÅ±veletek lassÃºak, sok regiszterre van szÃ¼ksÃ©gÃ¼nk, hogy egy beolvasott szÃ³ mindig a regiszterben maradhasson, amÃg szÃ¼ksÃ©g van rÃ¡.

RISC Reduced Instruction Set Computer â CsÃ¶kkentett utasÃtÃ¡skÃ©szletÅ± szÃ¡mÃtÃ³gÃ©p

A mikroprocesszorok lÃ©trejÃ¶ttÃ©t kÃ¶vetÅen kÃ©t irÃ¡nyzat alakult ki. â RISC, CISC Azt a szempontot tartottÃ¡k szem elÅtt, hogy a processzor kevÃ©s alapvetÅ utasÃtÃ¡st tudjon vÃ©grehajtani, de azokat rendkÃvÃ¼l gyorsan (jellemzÅen 1 Ã³rajelciklus alatt). Ezek a RISC (Reduced Instruction Set Computer - redukÃ¡lt utasÃtÃ¡skÃ©szletÅ±) processzorok. Itt az Ã¶sszetettebb funkciÃ³kat tÃ¶bb utasÃtÃ¡s kombinÃ¡ciÃ³jÃ¡val lehet megvalÃ³sÃtani. A RISC mikroprocesszorokba szÃ¡mos belsÅ regiszter kerÃ¼l integrÃ¡lÃ¡sra, ezÃ¡ltal is csÃ¶kkentve a memÃ³riÃ¡hoz valÃ³ fordulÃ¡s gyakorisÃ¡gÃ¡t Ã©s gyorsÃtva a mÅ± kÃ¶dÃ©st. Ugyancsak sajÃ¡tja ezen processzoroknak a - kÃ©sÅbb ismertetett - Ãºn. pipeline architektÃºra. Ennek lÃ©nyege az, hogy a mÅ±veleteket rÃ©szmÅ±veletekkÃ© bontjÃ¡k szÃ©t, Ã©s e rÃ©szmÅ±veleteket idÅben pÃ¡rhuzamosÃtjÃ¡k, A RISC processzorok az utolsÃ³ 10 Ã©vben - elsÅ sorban a nagyobb teljesÃtmÃ©nyt igÃ©nylÅ rendszereknÃ©l (pl. munkaÃ¡llomÃ¡sok) nyertek teret Nagyon kevÃ©s utasÃtÃ¡ssal rendelkeznek, tipikusan 50 kÃ¶rÃ¼l. Az adatÃºt egyszeri bejÃ¡rÃ¡sÃ¡val vÃ©grehajthatÃ³k ezek az utasÃtÃ¡sok, tehÃ¡t egy Ã³rajel alatt. Nem hasznÃ¡l mikroprogram interpretÃ¡lÃ¡st, ezÃ©rt sokkal gyorsabb, mint a CISC.

PÃ©lda: IBM 801, UltraSPARC, ARM

CISC Complex Instruction Set Computer â Ãsszetett utasÃtÃ¡skÃ©szletÅ± szÃ¡mÃtÃ³gÃ©p

Azok a processzorok tartoznak ide, amelyek utasÃtÃ¡skÃ©szlete lehetÅleg minden programozÃ³i igÃ©nyt ki prÃ³bÃ¡l elÃ©gÃteni, vagyis komplex utasÃtÃ¡skÃ©szletet alkot. Ezeket nevezzÃ¼k CISC (Complex Instruction Set Computer = komplex utasÃtÃ¡skÃ©szletÅ± szÃ¡mÃtÃ³gÃ©p) processzoroknak. MarkÃ¡ns elemei az Intel processzorok. A CISC tÃ¶rekvÃ©snek az egyik mozgatÃ³rugÃ³ja, hogy megprÃ³bÃ¡ljÃ¡k a magasabb szintÅ± nyelvek lehetÅsÃ©geit kÃ¶zelÃteni, vagyis, hogy a programozÃ¡s "munkaigÃ©nyes" alacsony szintjÃ©t, gÃ©pkÃ¶zeli voltÃ¡t Ãgy is ellensÃºlyozzÃ¡k. InterpretÃ¡lÃ¡st hasznÃ¡l, ezÃ©rt sokkal Ã¶sszetettebb utasÃtÃ¡sai vannak, mint egy RISC gÃ©pnek. TÃ¶bb szÃ¡z ilyen utasÃtÃ¡sa lehet. Az interpretÃ¡lÃ¡s miatt lassabb a vÃ©grehajtÃ¡s.

PÃ©lda: x86 architektÃºrÃ¡k pl. Intel 80x86 csalÃ¡d.# 16. SzÃ¡mÃtÃ³gÃ©p perifÃ©riÃ¡k: MÃ¡gneses Ã©s optikai adattÃ¡rolÃ¡s alapelvei, mÅ±kÃ¶dÃ©sÃ¼k (merevlemez, Audio CD, CD-ROM, CD-R, CD-RW, DVD, Bluray). SCSI, RAID. NyomtatÃ³k, egÃ©r, billentyÅ±zet. TelekommunikÃ¡ciÃ³s berendezÃ©sek (modem, ADSL, KÃ¡belTV-s internet)

SzÃ¡mÃtÃ³gÃ©p perifÃ©riÃ¡k

A szÃ¡mÃtÃ³gÃ©phez kÃ¼lÃ¶nbÃ¶zÅ perifÃ©riÃ¡k kapcsolhatÃ³k, melyek segÃtsÃ©gÃ©vel a felhasznÃ¡lÃ³k kommunikÃ¡lni tudnak a gazdagÃ©ppel. Ezek egy rÃ©sze beviteli, vagy kiviteli eszkÃ¶z, - amely az adatok bevitelÃ©re, vagy kiÃrÃ¡sÃ¡ra szolgÃ¡l. A hÃ¡ttÃ©rtÃ¡rolÃ³k feladata az adatok Ã©s programok hosszabb ideig tartÃ³ tÃ¡rolÃ¡sa. Tartalmuk a szÃ¡mÃtÃ³gÃ©p kikapcsolÃ¡sa utÃ¡n is megmarad. A fogalmat Ã¡ltalÃ¡ban azokra az eszkÃ¶zÃ¶kre alkalmazzÃ¡k, melyek kÃ¼lsÅleg csatlakoznak a gazdagÃ©phez, tipikusan egy szÃ¡mÃtÃ³gÃ©pes buszon keresztÃ¼l, mint pÃ©ldÃ¡ul az USB.

CsoportosÃtÃ¡suk:

- bemeneti perifÃ©riÃ¡k
- kimeneti perifÃ©riÃ¡k

MÃ¡gneses adattÃ¡rolÃ¡s alapelvei, mÅ±kÃ¶dÃ©se

Egy mÃ¡gneslemez egy vagy tÃ¶bb mÃ¡gnesezhetÅ bevonattal ellÃ¡tott alumÃniumkorongbÃ³l Ã¡ll. Egy indukciÃ³s fej lebeg a lemez felszÃne felett egy vÃ©kony lÃ©gpÃ¡rnÃ¡n Ha pozitÃv vagy negatÃv Ã¡ram folyik az indukciÃ³s tekercsben, a fej alatt a lemez magnetizÃ¡lÃ³dik, Ã©s ahogy a korong forog a fej alatt, Ãgy bitsorozatokat lehet felÃrni Amikor a fej egy mÃ¡gnesezett terÃ¼let felett halad Ã¡t, akkor pozitÃv vagy negatÃv Ã¡ram indukÃ¡lÃ³dik benne, Ãgy a korÃ¡bban eltÃ¡rolt biteket vissza lehet olvasni. Egy teljes kÃ¶rÃ¼fordulÃ¡s alatt felÃrt bitsorozat a sÃ¡v. Minden sÃ¡v rÃ¶gzÃtett mÃ©retÅ± tipikusan 512 bÃ¡jt mÃ©retÅ± szektorokra van osztva, melyeket egy fejlÃ©c elÅz meg, lehetÅvÃ© tÃ©ve a fej szinkronizÃ¡lÃ¡sÃ¡t ÃrÃ¡s Ã©s olvasÃ¡s elÅtt. Az adatok utÃ¡n hibajavÃtÃ³ kÃ³d helyezkedik el (Hamming vagy Reed-Solomon).

Minden lemeznek vannak mozgathatÃ³ karjai, melyek a forgÃ¡stengelytÅl sugÃ¡rirÃ¡nyban ki-be tudnak mozogni. Minden sugÃ¡rirÃ¡nyÃº pozÃciÃ³n egy-egy sÃ¡v ÃrhatÃ³ fel. TehÃ¡t a sÃ¡vok forgÃ¡stengely kÃ¶zÃ©ppontÃº koncentrikus kÃ¶rÃ¶k.

Egy lemezegysÃ©g tÃ¶bb, egymÃ¡s felett elhelyezett korongbÃ³l Ã¡ll. Minden felÃ¼lethez tartozik egy fej Ã©s egy mozgatÃ³kar. A karok rÃ¶gzÃtve vannak egymÃ¡shoz, Ãgy a fejek mindig ugyanarra a sugÃ¡rirÃ¡nyÃº pozÃciÃ³ra Ã¡llnak be. Egy adott sugÃ¡rirÃ¡nyÃº pozÃciÃ³hoz tartozÃ³ sÃ¡vok Ã¶sszessÃ©gÃ©t cilindereknek nevezzÃ¼k. ÃltalÃ¡ban 6-12 korong talÃ¡lhatÃ³ egymÃ¡s felett.

Egy szektor beolvasÃ¡sÃ¡hoz vagy kiÃrÃ¡sÃ¡hoz elÅszÃ¶r a fejet a megfelelÅ sugÃ¡rirÃ¡nyÃº pozÃciÃ³ba kell Ã¡llÃtani, ezt keresÃ©snek (seek) hÃvjÃ¡k. A fej kÃvÃ¡nt sugÃ¡rirÃ¡nyÃº pozÃciÃ³ba valÃ³ beÃ¡llÃ¡sa utÃ¡n van egy kis szÃ¼net, az Ãºn. forgÃ¡si kÃ©sleltetÃ©s, amÃg a keresett szektor a fej alÃ¡ fordul. A kÃ¼lsÅ sÃ¡vok hosszabbak, mint a belsÅk, a lemezek pedig a fejek pozÃciÃ³jÃ¡tÃ³l fÃ¼ggetlenÃ¼l Ã¡llandÃ³ szÃ¶gsebessÃ©ggel forognak, ezÃ©rt ez problÃ©mÃ¡t vet fel. MegoldÃ¡skÃ©pp a cilindereket zÃ³nÃ¡kba osztjÃ¡k, Ã©s a kÃ¼lsÅ zÃ³nÃ¡kban tÃ¶bb szektort tesznek egy sÃ¡vba. Minden szektor mÃ©rete egyforma. Minden lemezhez tartozik egy lemezvezÃ©rlÅ, egy lapka, amely vezÃ©rli a meghajtÃ³t.

Optikai adattÃ¡rolÃ¡s alapelvei, mÅ±kÃ¶dÃ©se

Az optikai adattÃ¡rolÃ³k megjelenÃ©se kÃ¶r alakÃº lemez, amelyek felÃ¼letÃ©n helyezkedik el az adattÃ¡rolÃ¡sra alkalmas rÃ©teg. A lemezek ÃrÃ¡sa Ã©s olvasÃ¡sa a nevÃ¼kbÅl adÃ³dÃ³an optikai eljÃ¡rÃ¡ssal tÃ¶r-tÃ©nik. Az optikai ÃrÃ¡s Ã©s az olvasÃ¡s lÃ©zersugÃ¡rral tÃ¶rtÃ©nik a lemez forgatÃ¡sa kÃ¶zben. A lemezen tÃ¶rtÃ©nÅ adatrÃ¶gzÃtÃ©skor a lÃ©zersugÃ¡r aprÃ³ mÃ©lyedÃ©seket hoz lÃ©tre spirÃ¡l alakÃº vonalban, Ãgy tÃ¡rolva a digitÃ¡lis adatot; az adat kiolvasÃ¡sÃ¡hoz ugyanilyen hullÃ¡mhosszÃº lÃ©zersugÃ¡r halad vÃ©gig a mÃ©lyedÃ©sek sorozatÃ¡n Ã©s olvassa vissza a digitÃ¡lis adatot aszerint, hogy a sugÃ¡r visszatÃ¼krÃ¶zÅdik, vagy szÃ©tszÃ³rÃ³dik a lemez felÃ¼letÃ©rÅl. Az optikai tÃ¡rolÃ³kat tÃ¶bb tulajdonsÃ¡guk is jelentÅsen megkÃ¼lÃ¶nbÃ¶zteti a mÃ¡gneses tÃ¡raktÃ³l: az optikai tÃ¡rolÃ¡s elmÃ©letben sokkal nagyobb adatsÅ±rÅ±sÃ©get enged meg, mivel a fÃ©ny sokkal kisebb terÃ¼letre fÃ³kuszÃ¡lhatÃ³, mint a mÃ¡gneses adattÃ¡rolÃ³kban az elemi mÃ¡gnesezhetÅ rÃ©szecskÃ©k mÃ©rete. TovÃ¡bbÃ¡, a megfelelÅ minÅsÃ©gÅ± Ã©s megfelelÅen kezelt optikai lemezek Ã©lettartama Ã©vszÃ¡zadokban mÃ©rhetÅ, ezenkÃvÃ¼l nem Ã©rzÃ©kenyek az elektromÃ¡gneses behatÃ¡sokra sem.

A felÃ¼leten elhelyezkedÅ mÃ©lyedÃ©seket Ã¼regnek (pit), az Ã¼regek kÃ¶zÃ¶tti Ã©rintetlen terÃ¼leteket pedig szintnek (land) hÃvjÃ¡k.

Az tÅ±nik a legegyszerÅ±bbnek, hogy Ã¼reget hasznÃ¡ljunk a 0, szintet az 1 tÃ¡rolÃ¡sÃ¡hoz, ennÃ©l azonban megbÃzhatÃ³bb, ha az Ã¼reg/szint vagy a szint/Ã¼reg Ã¡tmenetet hasznÃ¡ljuk az 1-hez, az Ã¡tmenet hiÃ¡nyÃ¡t pedig a 0-hoz, ezÃ©rt ez utÃ³bbi mÃ³dszert alkalmazzÃ¡k.

Merevlemez (HDD)

- MÃ¡gneses adattÃ¡rolÃ³
- TÃ¡rolÃ³kapacitÃ¡s: 500 GB â 12 TB
- ÃrÃ¡sa Ã©s olvasÃ¡si sebessÃ©g: fÃ¼gg a forgÃ¡si sebessÃ©gtÅl, ez jellemzÅen 5400, 7200, 1000 vagy 15000 fordulat/perc, Ã©s az adatsÅ±rÅ±sÃ©gtÅl (egy adathordozÃ³ fizikai felÃ¼letÃ©vel arÃ¡nyos tÃ¡rolÃ³kapacitÃ¡sa)

Audio CD - A jel sÅ±rÅ±sÃ©ge Ã¡llandÃ³ a spirÃ¡l mentÃ©n - 74 percnyi anyag fÃ©r rÃ¡ (Beethoven IX. szimfÃ³niÃ¡ja kiadhatÃ³ legyen) - ÃllandÃ³ kerÃ¼leti sebessÃ©g, ehhez szÃ¼ksÃ©ges a vÃ¡ltozÃ³ forgÃ¡si sebessÃ©g (120 cm/mp) - Nincs hibajavÃtÃ¡s, mivel nem gond, ha nÃ©hÃ¡ny bit elvÃ©sz az audio anyagbÃ³l

CD-ROM - UniverzÃ¡lis adathordozÃ³, illetve mÃ©dialemez. - Csak olvashatÃ³ (vÃ©glegesÃtett) adathordozÃ³. - NÃ©pszerÅ±en hasznÃ¡ltÃ¡k szoftverek Ã©s adatok terjesztÃ©sÃ©re - Az ilyen tÃpusÃº lemezeket kereskedelmi forgalomban hozzÃ¡k lÃ©tre, Ã©s lÃ©trehozÃ¡suk utÃ¡n nem menthet rÃ¡juk adatokat. - 650 MB tÃ¡rolhatÃ³

CD-R - ÃrÃ³ berendezÃ©ssel rÃ¶gzÃthetÅ az adat (1x) - ÃjdonsÃ¡g: â¦ ÃrÃ³ lÃ©zernyalÃ¡b â¦ AlumÃnium helyett arany felÃ¼let â¦ Ãregek Ã©s szintek helyett festÃ©krÃ©teg alkalmazÃ¡sa: Kezdetben Ã¡tlÃ¡tszÃ³ a festÃ©krÃ©teg (cianin (zÃ¶ld) vagy ftalocianin (sÃ¡rgÃ¡s)) - 700 MB tÃ¡rolhatÃ³

CD-RW - ÃjraÃrhatÃ³ optikai lemez - A CD-RW lemez adatait szÃ¡mos alkalommal tÃ¶rÃ¶lhetjÃ¼k Ã©s rÃ¶gzÃthetjÃ¼k. - ÃjdonsÃ¡g: â¦ MÃ¡s adattÃ¡rolÃ³ rÃ©teg: âª EzÃ¼st, indium, antimon Ã©s tellÃºr Ã¶tvÃ¶zet âª KÃ©tfÃ©le stabil Ã¡llapot: kristÃ¡lyos Ã©s amorf (mÃ¡s fÃ©nyvisszaverÅ kÃ©pessÃ©g) - 3 eltÃ©rÅ energiÃ¡jÃº lÃ©zer: âª Legmagasabb energia: megolvad az Ã¶tvÃ¶zet â amorf âª KÃ¶zepes energia: megolvad â kristÃ¡lyos Ã¡llapot âª Alacsony energia: anyag Ã¡llapotnak Ã©rzÃ©kelÃ©se, de meg nem vÃ¡ltozik

DVD - Nagy kapacitÃ¡sÃº optikai tÃ¡rolÃ³, amely leginkÃ¡bb mozgÃ³kÃ©p Ã©s jÃ³ minÅsÃ©gÅ± hang, valamint adat tÃ¡rolÃ¡sÃ¡ra hasznÃ¡latos - MÃ©reteit tekintve Ã¡ltalÃ¡ban akkora, mint a CD, vagyis 120 mm Ã¡tmÃ©rÅjÅ±. - LÃ©tezik egyrÃ©tegÅ±/kÃ©trÃ©tegÅ± illetve egyoldalas/kÃ©toldalas lemez (4,5 GB â 17 GB) - Nagyobb jelsÅ±rÅ±sÃ©g, mert â¦ Kisebbek az Ã¼regek (0,4 Î¼m (CD: 0,8 Î¼m)) â¦ Szorosabb spirÃ¡lok â¦ VÃ¶rÃ¶s lÃ©zert hasznÃ¡ltak

Blu-Ray - A DVD technolÃ³gia tovÃ¡bbfejlesztÃ©se, a Blu-Ray disc - KÃ©k lÃ©zer hasznÃ¡lata ÃrÃ¡sra Ã©s olvasÃ¡sra a vÃ¶rÃ¶s helyett â¦ RÃ¶videbb hullÃ¡mhossz, jobban fÃ³kuszÃ¡lhatÃ³, kisebb mÃ©lyedÃ©sek - 25 GB (egyoldalas) Ã©s 50 GB (kÃ©toldalas) adattÃ¡rolÃ¡si kÃ©pessÃ©g

SCSI, RAID

SCSI

Az SCSI-lemezek nem kÃ¼lÃ¶nbÃ¶znek az IDE-lemezektÅl abban a tekintetben, hogy ezek is cilinderekre, sÃ¡vokra Ã©s szektorokra vannak osztva, de mÃ¡s az interfÃ©szÃ¼k, Ã©s sokkal nagyobb az adatÃ¡tviteli sebessÃ©gÃ¼k. Az 5 MHz-estÅl a 160 MHz-ig nagyon sok vÃ¡ltozatot kifejlesztettek.

A SCSI tÃ¶bb egy merevlemez-interfÃ©sznÃ©l. Ez egy sÃn, amelyre egy SCSI-vezÃ©rlÅ Ã©s legfeljebb hÃ©t eszkÃ¶z csatlakoztathatÃ³. Ezek kÃ¶zÃ¶tt lehet egy vagy tÃ¶bb SCSI-merevlemez, CD-ROM, CD-ÃrÃ³, szkenner, szalagegysÃ©g Ã©s mÃ¡s SCSI-perifÃ©ria.

A SCSI-vezÃ©rlÅk Ã©s âperifÃ©riÃ¡k kezdemÃ©nyezÅ Ã©s fogadÃ³ Ã¼zemmÃ³dban mÅ±kÃ¶dhetnek. ÃltalÃ¡ban a kezdemÃ©nyezÅkÃ©nt mÅ±kÃ¶dÅ vezÃ©rlÅ adja ki a parancsokat a fogadÃ³kÃ©nt viselkedÅ lemezegysÃ©geknek Ã©s egyÃ©b perifÃ©riÃ¡knak.

A szabvÃ¡ny megengedi, hogy az Ã¶sszes eszkÃ¶z egyszerre mÅ±kÃ¶djÃ¶n, Ãgy nagyban nÃ¶velhetÅ a hatÃ©konysÃ¡g tÃ¶bb folyamatot futtatÃ³ kÃ¶rnyezetben.

RAID

A RAID tÃ¡rolÃ¡si technolÃ³gia, mely segÃtsÃ©gÃ©vel az adatok elosztÃ¡sa vagy replikÃ¡lÃ¡sa tÃ¶bb fizikailag fÃ¼ggetlen merevlemezen, egy logikai lemez lÃ©trehozÃ¡sÃ¡val lehetsÃ©ges. Minden RAID szint alapjÃ¡ban vÃ©ve vagy az adatbiztonsÃ¡g nÃ¶velÃ©sÃ©t vagy az adatÃ¡tviteli sebessÃ©g nÃ¶velÃ©sÃ©t szolgÃ¡lja.

Azon tÃºl, hogy a RAID szoftverszempontbÃ³l egyetlen lemeznek lÃ¡tszik, az adatok szÃ©t vannak osztva a meghajtÃ³k kÃ¶zÃ¶tt, lehetÅvÃ© tÃ©ve a pÃ¡rhuzamos mÅ±kÃ¶dÃ©st.

A RAID alapÃ¶tlete a lemezegysÃ©gek csÃkokra (stripes) bontÃ¡sa. Ezek a csÃkok azonban nem azonosak a lemez fizikai sÃ¡vjaival.

RAID-0 (Ã¶sszefÅ±zÃ©s vagy csÃkozÃ¡s)

Lemezek egyszerÅ± Ã¶sszefÅ±zÃ©sÃ©t jelenti, viszont semmilyen redundanciÃ¡t nem ad, Ãgy nem biztosÃt hibatÅ±rÃ©st, azaz egyetlen meghajtÃ³ meghibÃ¡sodÃ¡sa az egÃ©sz tÃ¶mb hibÃ¡jÃ¡t okozza. Mind az ÃrÃ¡si, mind az olvasÃ¡si mÅ±veletek pÃ¡rhuzamosÃtva tÃ¶rtÃ©nnek, ideÃ¡lis esetben a sebessÃ©g az egyes lemezek sebessÃ©gÃ©nek Ã¶sszege lesz, Ãgy a mÃ³dszer a RAID szintek kÃ¶zÃ¼l a legjobb teljesÃtmÃ©nyt nyÃºjtja (a tÃ¶bbi mÃ³dszernÃ©l a redundancia kezelÃ©se lassÃtja a rendszert)

RAID-1 (tÃ¼krÃ¶zÃ©s)

A RAID 1 eljÃ¡rÃ¡s alapja az adatok tÃ¼krÃ¶zÃ©se (disk mirroring), azaz az informÃ¡ciÃ³k egyidejÅ± tÃ¡rolÃ¡sa a tÃ¶mb minden elemÃ©n. Az adatok olvasÃ¡sa pÃ¡rhuzamosan tÃ¶rtÃ©nik a diszkekrÅl, felgyorsÃtvÃ¡n az olvasÃ¡s sebessÃ©gÃ©t; az ÃrÃ¡s normÃ¡l sebessÃ©ggel, pÃ¡rhuzamosan tÃ¶rtÃ©nik a meghajtÃ³kon. Az eljÃ¡rÃ¡s igen jÃ³ hibavÃ©delmet biztosÃt, bÃ¡rmely meghajtÃ³ meghibÃ¡sodÃ¡sa esetÃ©n folytatÃ³dhat a mÅ±kÃ¶dÃ©s.

RAID-2

Egyes meghajtÃ³kat hibajavÃtÃ³ tÃ¡rolÃ¡sÃ¡ra tartanak fenn. A hibajavÃtÃ³ kÃ³d lÃ©nyege, hogy az adatbitekbÅl valamilyen matematikai mÅ±velet segÃtsÃ©gÃ©vel redundÃ¡ns biteket kÃ©peznek. A hasznÃ¡lt eljÃ¡rÃ¡stÃ³l fÃ¼ggÅen a kapott kÃ³d akÃ¡r tÃ¶bb bithiba Ã©szlelÃ©sÃ©re, illetve javÃtÃ¡sÃ¡ra alkalmas. A vÃ©delem Ã¡ra a megnÃ¶vekedett adatmennyisÃ©g. A mÃ³dszer esetleges lemezhiba esetÃ©n kÃ©pes annak detektÃ¡lÃ¡sÃ¡ra, illetve kijavÃtÃ¡sÃ¡ra

RAID-3

A RAID 3 felÃ©pÃtÃ©se hasonlÃt a RAID 2-re, viszont nem a teljes hibajavÃtÃ³ kÃ³d, hanem csak egy lemeznyi paritÃ¡sinformÃ¡ciÃ³ tÃ¡rolÃ³dik. Egy adott paritÃ¡scsÃk a kÃ¼lÃ¶nbÃ¶zÅ lemezeken azonos pozÃciÃ³ban elhelyezkedÅ csÃkokbÃ³l XOR mÅ±velet segÃtsÃ©gÃ©vel kaphatÃ³ meg. A rendszerben egy meghajtÃ³ kiesÃ©se nem okoz problÃ©mÃ¡t, mivel a rajta lÃ©vÅ informÃ¡ciÃ³ a tÃ¶bbi meghajtÃ³ (a paritÃ¡st tÃ¡rolÃ³ meghajtÃ³t is beleÃ©rtve) XOR-akÃ©nt megkaphatÃ³.

RAID-4

A RAID 4 felÃ©pÃtÃ©se a RAID 3-mal megegyezik. Az egyetlen kÃ¼lÃ¶nbsÃ©g, hogy itt nagymÃ©retÅ± csÃkokat definiÃ¡lnak, Ãgy egy rekord egy meghajtÃ³n helyezkedik el, lehetÅvÃ© tÃ©ve egyszerre tÃ¶bb (kÃ¼lÃ¶nbÃ¶zÅ meghajtÃ³kon elhelyezkedÅ) rekord pÃ¡rhuzamos ÃrÃ¡sÃ¡t, illetve olvasÃ¡sÃ¡t (multi-user mode). ProblÃ©mÃ¡t okoz viszont, hogy a paritÃ¡s-meghajtÃ³ adott csÃkjÃ¡t minden egyes ÃrÃ¡skor frissÃteni kell (plusz egy olvasÃ¡s Ã©s ÃrÃ¡s), aminek kÃ¶vetkeztÃ©ben pÃ¡rhuzamos ÃrÃ¡skor a paritÃ¡smeghajtÃ³ a rendszer szÅ±k keresztmetszetÃ©vÃ© vÃ¡lik.

RAID-5

A RAID 5 a paritÃ¡s informÃ¡ciÃ³t nem egy kitÃ¼ntetett meghajtÃ³n, hanem âkÃ¶rbeforgÃ³ paritÃ¡sâ (rotating parity) hasznÃ¡latÃ¡val, egyenletesen az Ã¶sszes meghajtÃ³n elosztva tÃ¡rolja, kikÃ¼szÃ¶bÃ¶lvÃ©n a paritÃ¡s-meghajtÃ³ jelentette szÅ±k keresztmetszetet. Mind az ÃrÃ¡si, mind az olvasÃ¡si mÅ±veletek pÃ¡rhuzamosan vÃ©gezhetÅek. Egy meghajtÃ³ meghibÃ¡sodÃ¡sa esetÃ©n az adatok sÃ©rtetlenÃ¼l visszaolvashatÃ³ak, a hibÃ¡s meghajtÃ³ adatait a vezÃ©rlÅ a tÃ¶bbi meghajtÃ³rÃ³l ki tudja szÃ¡molni.

NyomtatÃ³k, egÃ©r, billentyÅ±zet

NyomtatÃ³k

MÃ¡trixnyomtatÃ³k

A nyomtatÃ³fejben aprÃ³ tÅ±k vannak (Ã¡ltalÃ¡ban 9 vagy 24 db). A papÃr elÅtt egy kifeszÃtett festÃ©kszalag mozog, amelyre a tÅ±k rÃ¡Ã¼tnek, Ã©s lÃ©trehoznak a papÃron egy pontot. A kÃ©p ezekbÅl a pontokbÃ³l fog Ã¡llni. A tÅ±ket elektromÃ¡gneses tÃ©r mozgatja, Ã©s rugÃ³erÅ hÃºzza vissza eredeti helyÃ¼kre. Ezzel az eljÃ¡rÃ¡ssal nem csak karakterek, hanem kÃ©pek, rajzok is nyomtathatÃ³ak. A nyomtatott kÃ©pek felbontÃ¡sa gyenge, a nyomtatÃ³ lassÃº viszont olcsÃ³k Ã©s nagyon megbÃzhatÃ³k.

Tintasugaras nyomtatÃ³:

A tintasugaras nyomtatÃ³k tintapatronok segÃtsÃ©gÃ©vel tintacseppeket juttatnak a papÃrlapra. A patronban van egy porlasztÃ³, ez megfelelÅ mÃ©retÅ± tintacseppekre alakÃtja a tintÃ¡t, Ã©s a papÃrlapra juttatja azt. A szÃnes tintasugaras nyomtatÃ³ szÃnes tintapatronokat hasznÃ¡l, Ã¡ltalÃ¡ban nÃ©gy alapszÃn hasznÃ¡latÃ¡val keveri ki a megfelelÅ Ã¡rnyalatokat: ciÃ¡nkÃ©k, bÃborvÃ¶rÃ¶s, sÃ¡rga Ã©s fekete szÃnek hasznÃ¡latÃ¡val. Minden tintasugaras nyomtatÃ³ porlasztÃ¡ssal juttatja a tintacseppeket a papÃrlapra, de a porlasztÃ¡s mÃ³dszere vÃ¡ltozÃ³. Ez tÃ¶rtÃ©nhet piezoelektromos Ãºton, elektrosztatikusan, vagy gÅzbuborÃ©kok segÃtsÃ©gÃ©vel.

A gÅzbuborÃ©kos nyomtatÃ³ a kÃ¶vetkezÅ mÃ³don mÅ±kÃ¶dik: A nyomtatÃ³ cserÃ©lhetÅ tintapatronja a papÃr felett oldalirÃ¡nyban mozog. A nyomtatÃ³fejben lÃ©vÅ, tintÃ¡val tÃ¶ltÃ¶tt kamrÃ¡cskÃ¡khoz szabad szemmel alig lÃ¡thatÃ³ fÃºvÃ³kÃ¡k (porlasztÃ³k) kapcsolÃ³dnak. Azokat a kamrÃ¡kat, mely a nyomtatandÃ³ kÃ©prÃ©szlet soron kÃ¶vetkezÅ kÃ©ppontjÃ¡hoz szÃ¼ksÃ©gesek, elektromos impulzus melegÃti fel, minek kÃ¶vetkeztÃ©ben a tinta a melegÃtÃ©si helyeken felforr, Ã©s a keletkezÅ gÅzbuborÃ©k egy-egy tintacseppet lÅ a porlasztÃ³kon keresztÃ¼l a papÃrlapra. A tintasugaras nyomtatÃ³k egy-egy karaktert sokkal tÃ¶bb kÃ©ppontbÃ³l Ã¡llÃtanak Ã¶ssze mint pÃ©ldÃ¡ul a mÃ¡trixnyomtatÃ³k, ezÃ©rt sokkal szebb kÃ©pet is adnak annÃ¡l: megfelelÅ tintasugaras nyomtatÃ³val igen jÃ³ minÅsÃ©gÅ±, szÃnes kÃ©pek, akÃ¡r fotÃ³k is nyomtathatÃ³k.

LÃ©zernyomtatÃ³

A nyomtatÃ³ szÃve egy fÃ©nyÃ©rzÃ©keny anyaggal bevont forgÃ³ henger. Egy-egy lap nyomtatÃ¡sa elÅtt eletromosan feltÃ¶ltÅdik. Ezt kÃ¶vetÅen egy lÃ©zer fÃ©nye pÃ¡sztÃ¡zza vÃ©gig a hengert hosszÃ¡ban, amelyet egy nyolcszÃ¶gletÅ± tÃ¼kÃ¶rrel irÃ¡nyÃtanak a hengerre. A fÃ©nyt modulÃ¡ljÃ¡k, hogy vilÃ¡gos Ã©s sÃ¶tÃ©t pontokat kapjanak. Azok a pontok, ahol fÃ©ny Ã©ri a hengert, elveszÃtik elektromos tÃ¶ltÃ©sÃ¼ket. MiutÃ¡n egy pontokbÃ³l Ã¡llÃ³ sor elkÃ©szÃ¼lt, a henger elfordul Ã©s elkezdÅdhet a kÃ¶vetkezÅ sor elÅÃ¡llÃtÃ¡sa. KÃ©sÅbb az elsÅ sor elÃ©ri a tonerkazettÃ¡t, amely elektrosztatikusan Ã©rzÃ©keny fekete port tartalmaz. A por hozzÃ¡tapad a mÃ©g feltÃ¶ltÃ¶tt pontokhoz, Ãgy lÃ¡thatÃ³vÃ¡ vÃ¡lik a sor. TovÃ¡bb fordulva a bevont henger hozzÃ¡nyomÃ³dik a papÃrhoz, Ã¡tadva a papÃrnak a festÃ©ket. A papÃr ezutÃ¡n felmelegÃtett gÃ¶rgÅk kÃ¶zÃ¶tt halad el, ezÃ¡ltal a festÃ©k vÃ©glegesen hozzÃ¡tapad a papÃrhoz. A lÃ©zernyomtatÃ³k kivÃ¡lÃ³ minÅsÃ©gÅ± kÃ©pet kÃ©szÃtenek, nagy a rugalmassÃ¡guk, sebessÃ©gÃ¼k Ã©s elfogadhatÃ³ a kÃ¶ltsÃ©g.

EgÃ©r

Az egÃ©r egy grafikus felÃ¼leten valÃ³ mutatÃ³ mozgatÃ¡sÃ¡ra szolgÃ¡lÃ³ perifÃ©ria. Az egÃ©ren egy, kettÅ vagy akÃ¡r tÃ¶bb nyomÃ³gomb van, illetve egy gÃ¶rgÅ is lehet rajta. BelsejÃ©ben talÃ¡lhatÃ³ Ã©rzÃ©kelÅ felismeri Ã©s tovÃ¡bbÃtja a szÃ¡mÃtÃ³gÃ©p felÃ© az egÃ©r mozgÃ¡sÃ¡t egy sima felÃ¼leten Optikai Az optikai egÃ©r a mozgÃ¡sokat egy optikai szenzor segÃtsÃ©gÃ©vel ismerte fel, mely egy fÃ©nykibocsÃ¡tÃ³ diÃ³dÃ¡t hasznÃ¡lt a megvilÃ¡gÃtÃ¡shoz. Az elsÅ optikai egereket csak egy speciÃ¡lis fÃ©mes egÃ©rpadon lehetett hasznÃ¡lni, melyre kÃ©k Ã©s szÃ¼rke vonalak hÃ¡lÃ³ja volt felfestve. MiutÃ¡n a szÃ¡mÃtÃ³gÃ©pes eszkÃ¶zÃ¶k egyre olcsÃ³bbak lettek, lehetÅsÃ©g nyÃlt egy sokkal pontosabb kÃ©pelemzÅ chip beÃ©pÃtÃ©sÃ©re is az egÃ©rbe, melynek segÃtsÃ©gÃ©vel az egÃ©r mozgÃ¡sÃ¡t mÃ¡r szinte bÃ¡rmilyen felÃ¼leten Ã©rzÃ©kelni lehetett, Ãgy tÃ¶bbÃ© nem volt szÃ¼ksÃ©g speciÃ¡lis egÃ©rpadra. Ez a fejlesztÃ©s megnyitotta a lehetÅsÃ©get az optikai egerek elterjedÃ©se elÅtt. A modern optikai egerek egy reflexszenzor segÃtsÃ©gÃ©vel sorozatos kÃ©peket kÃ©szÃtenek az egÃ©r alatti terÃ¼letrÅl. A kÃ©pek kÃ¶zÃ¶tti eltÃ©rÃ©st egy kÃ©pelemzÅ chip dolgozza fel, Ã©s az eredmÃ©nyt a kÃ©t tengelyhez viszonyÃtott elmozdulÃ¡ssÃ¡ alakÃtja.

Mechanikus Egy golyÃ³ kÃ©t egymÃ¡shoz kÃ©pest 90 fokban elhelyezett tengelyt forgat, melyek tovÃ¡bbÃtjÃ¡k a mozgÃ¡sÃ¡t a fÃ©nyÃ¡teresztÅ rÃ©sekkel rendelkezÅ korongoknak. Az optocsatolÃ³k infravÃ¶rÃ¶s LEDjei Ã¡tvilÃ¡gÃtanak a hozzÃ¡juk tartozÃ³ korongok rÃ©sein. BÃ¡rmely korong elfordulÃ¡sakor a rajta lÃ©vÅ rÃ©sek Ã¡tengedik LED fÃ©nyÃ©t, mÃg a rÃ©sek kÃ¶zÃ¶tti fogak nem. VÃ©geredmÃ©nyben az egÃ©r elmozdulÃ¡sa fÃ©nyimpulzusok sorozatÃ¡vÃ¡ vÃ¡ltozik, mÃ©gpedig annÃ¡l tÃ¶bb fÃ©nyimpulzus keletkezik, minÃ©l nagyobb az egÃ©r Ã¡ltal megtett Ãºt, A fÃ©nyÃ©rzÃ©keny szenzorok Ã©rzÃ©kelik a fÃ©nyimpulzusokat Ã©s elektromos jelekkÃ© alakÃtjÃ¡k.

BillentyÅ±zet

A billentyÅ±zet gombjai kÃ¡belezÃ©s szempontjÃ¡bÃ³l egy Ãºn. billentyÅ±zet-mÃ¡trixban vannak elhelyezve. Egy meghatÃ¡rozott billentyÅ± lenyomÃ¡sÃ¡nak vagy felengedÃ©sÃ©nek Ã©szlelÃ©sekor a belsÅ mikroprocesszor egy, az adott billentyÅ±t egyÃ©rtelmÅ±en azonosÃtÃ³ Ãºn. scan-kÃ³dot kÃ¼ld a szÃ¡mÃtÃ³gÃ©p felÃ©. Ugyanezen billentyÅ± felengedÃ©sekor a mikroprocesszor egy mÃ¡sik, felengedÃ©si scan-kÃ³dot tovÃ¡bbÃt a billentyÅ±zet-illesztÅ Ã¡ramkÃ¶r felÃ©. EzÃ¡ltal rÃ©szint kikÃ¼szÃ¶bÃ¶lhetÅ a tÃ¶bb billentyÅ± kÃ¶zel egyidejÅ± lenyomÃ¡sÃ¡bÃ³l adÃ³dÃ³ jelensÃ©g, a karakterek "elvesztÃ©se". A megfelelÅ gomb vagy kombinÃ¡ciÃ³k Ã©rtelmezÃ©se Ã©s feldolgozÃ¡sa Ãgy teljesen a szÃ¡mÃtÃ³gÃ©p billentyÅ±zetkezelÅ rutinjÃ¡nak feladata.

TelekommunikÃ¡ciÃ³s berendezÃ©sek

Modem

A modem egy olyan berendezÃ©s, ami egy vivÅhullÃ¡m modulÃ¡ciÃ³jÃ¡val a digitÃ¡lis jelet analÃ³g informÃ¡ciÃ³vÃ¡, illetve a mÃ¡sik oldalon ennek demodulÃ¡ciÃ³jÃ¡val Ãºjra digitÃ¡lis informÃ¡ciÃ³vÃ¡ alakÃtja. Az eljÃ¡rÃ¡s cÃ©lja, hogy a digitÃ¡lis adatot analÃ³g mÃ³don Ã¡tvihetÅvÃ© tegye. A modulÃ¡ciÃ³ kÃ¼lÃ¶nfÃ©le eljÃ¡rÃ¡sok csoportja, melyek biztosÃtjÃ¡k, hogy egy tipikusan szinuszos jel - a vivÅ - kÃ©pes legyen informÃ¡ciÃ³ hordozÃ¡sÃ¡ra. A szinuszos jel hÃ¡rom fÅ paramÃ©terÃ©t, az amplitÃºdÃ³jÃ¡t, a fÃ¡zisÃ¡t vagy a frekvenciÃ¡jÃ¡t mÃ³dosÃthatja a modulÃ¡ciÃ³s eljÃ¡rÃ¡s, azÃ©rt, hogy a vivÅ informÃ¡ciÃ³t hordozhasson. NÃ©hÃ¡ny ok, ami miatt szÃ¼ksÃ©ges a kÃ¶zvetÃtÅ kÃ¶zegen valÃ³ Ã¡tkÃ¼ldÃ©st megelÅzÅ modulÃ¡ciÃ³: A modem egy mÃ¡sik modemmel mÅ±kÃ¶dik pÃ¡rban, ezek az Ã¡tviteli kÃ¶zeg kÃ©t vÃ©gÃ©n vannak. SzigorÃº Ã©rtelemben vÃ©ve a kÃ©t modem kÃ©t adatÃ¡tviteli berendezÃ©st kÃ¶t Ã¶ssze, azonban a mÃ¡sik vÃ©gberendezÃ©s tovÃ¡bb csatlakozhat az internet felÃ©.

ADSL

Az ADSL vagyis az aszimmetrikus digitÃ¡lis elÅfizetÅi vonal valÃ³jÃ¡ban egy kommunikÃ¡ciÃ³s technolÃ³gia, amely egy csavart rÃ©zÃ©rpÃ¡rÃº telefonkÃ¡belen keresztÃ¼l juttat el adatot A pontbÃ³l B pontba. A technolÃ³gia segÃtsÃ©gÃ©vel a hagyomÃ¡nyos modemekhez kÃ©pest gyorsabb digitÃ¡lis adatÃ¡tvitel Ã©rhetÅ el, ezÃ©rt igazi Ã¡ttÃ¶rÃ©s volt megjelenÃ©se az internetszolgÃ¡ltatÃ¡s piacÃ¡n. Az ADSL jellemzÅje, hogy a letÃ¶ltÃ©si Ã©s a feltÃ¶ltÃ©si sÃ¡vszÃ©lessÃ©g arÃ¡nya nem egyenlÅ (vagyis a vonal aszimmetrikus), amely az otthoni felhasznÃ¡lÃ³knak kedvezve a letÃ¶ltÃ©s sebessÃ©gÃ©t helyezi elÅnybe a feltÃ¶ltÃ©ssel szemben. Mind technikai, mind Ã¼zleti okai vannak az ADSL gyors elterjedÃ©sÃ©nek. A technikai elÅnyt az adja, hogy a zajelnyomÃ¡si lehetÅsÃ©geket kihasznÃ¡lva lehetÅvÃ© teszi nagyobb tÃ¡volsÃ¡gon is a gyors adatÃ¡tvitelt a felhasznÃ¡lÃ³ lakÃ¡sa Ã©s a DSLAM eszkÃ¶z kÃ¶zÃ¶tt (amely a telefonkÃ¶zpontokban helyezkedik el).

KÃ¡belTV-s internet

A kÃ¡belszolgÃ¡ltatÃ³k minden vÃ¡rosban fÅ telephellyel rendelkeznek, valamint rengeteg, elektronikÃ¡val zsÃºfolt dobozzal szerte a mÅ±kÃ¶dÃ©si terÃ¼letÃ¼kÃ¶n, amelyeket fejÃ¡llomÃ¡soknak neveznek. A fejÃ¡llomÃ¡sok nagy sÃ¡vszÃ©lessÃ©gÅ± kÃ¡belekkel vagy Ã¼vegkÃ¡belekkel kapcsolÃ³dnak a fÅ telephelyhez. Minden fejÃ¡llomÃ¡srÃ³l egy vagy tÃ¶bb kÃ¡bel indul el, otthonok Ã©s irodÃ¡k szÃ¡zain halad keresztÃ¼l. Minden elÅfizetÅ a rajta keresztÃ¼lhaladÃ³ kÃ¡belhez csatlakozik. Ãgy a felhasznÃ¡lÃ³k osztoznak egy fejÃ¡llomÃ¡shoz vezetÅ kÃ¡belen, ezÃ©rt a kiszolgÃ¡lÃ¡s sebessÃ©ge attÃ³l fÃ¼gg, hogy pillanatnyilag hÃ¡nyan hasznÃ¡ljÃ¡k az adott vezetÃ©ket. A kÃ¡belek sÃ¡vszÃ©lessÃ©ge 750 MHz.# 13. SzÃ¡mÃtÃ³gÃ©p-hÃ¡lÃ³zati architektÃºrÃ¡k, szabvÃ¡nyosÃtÃ³k (ISO/OSI, Internet, ITU, IEEE)

ISO

International Organization for Standardization, NemzetkÃ¶zi SzabvÃ¡nyÃ¼gyi Szervezet

MindenfÃ©le szabvÃ¡nyokat adnak ki, 165 tagÃ¡llam nemzeti szabvÃ¡nyÃ¼gyi szervezete alkotja. A tÃ¡vkÃ¶zlÃ©si szabvÃ¡nyokhoz az ISO Ã©s az ITU-T gyakran egyÃ¼ttmÅ±kÃ¶dik, hogy a szabvÃ¡nyok kompatibilisek legyenek egymÃ¡ssal.

OSI

A szÃ¡mÃtÃ³gÃ©pek kommunikÃ¡ciÃ³jÃ¡hoz szÃ¼ksÃ©ges hÃ¡lÃ³zati protokollt hatÃ¡rozza meg.

OSI - Open System Interconnection

A kÃ¼lÃ¶nbÃ¶zÅ protokollok Ã¡ltal nyÃºjtott funkciÃ³kat rendezi egymÃ¡sra Ã©pÃ¼lÅ rÃ©tegekbe. Minden rÃ©teg csak az alsÃ³bb rÃ©tegek Ã¡ltal nyÃºjtott funkciÃ³kra tÃ¡maszkodhat, Ã©s az Ã¡ltala nyÃºjtott funkciÃ³kat csak a felette lÃ©vÅ rÃ©teg szÃ¡mÃ¡ra nyÃºjthatja. Ezt a rendszert gyakran protokoll veremnek is nevezik. Az OSI modell hÃ©t rÃ©teget definiÃ¡l, az alsÃ³bb rÃ©tegek azok, amelyeket hardver szinten is megvalÃ³sÃtanak, a felsÅbbek szoftveresen kerÃ¼lnek megvalÃ³sÃtÃ¡sra.

A rÃ©tegek alulrÃ³l felfelÃ©

- Fizikai rÃ©teg
  - feladata, hogy a biteket tovÃ¡bbÃtsa a kommunikÃ¡ciÃ³s csatornÃ¡n
  - mekkora feszÃ¼ltsÃ©g kell a 0, 1 bitek reprezentÃ¡lÃ¡sÃ¡hoz, mennyi idÅ, hogyan jÃ¶n lÃ©tre az Ã¶sszekÃ¶ttetÃ©s stb.
- Adatkapcsolati rÃ©teg

        - Ã¡tvitendÅ adatokat a kÃ¼ldÅ fÃ©l oldalÃ¡n adatkeretekbe tÃ¶rdeli, Ã©s sorrendben tovÃ¡bbÃtja
        - a fogadÃ³ fÃ©l nyugtÃ¡zza minden keret helyes vÃ©telÃ©t
        - forgalomszabÃ¡lyozÃ¡s, hibakezelÃ©s
- HÃ¡lÃ³zati rÃ©teg
    - milyen Ãºtvonalon kell a csomagokat a forrÃ¡sÃ¡llomÃ¡stÃ³l a cÃ©lig eljuttatni
    - lehet statikus, Ã©s dinamikus meghatÃ¡rozÃ¡s is
- SzÃ¡llÃtÃ¡si rÃ©teg
    - forgalomszabÃ¡lyozÃ¡s, hibajavÃtÃ¡s, multiplexelÃ©s
    - megbÃzhatÃ³sÃ¡g: pl ellenÅrzÅ Ã¶sszeggel megnÃ©zzÃ¼k, hogy az adat sÃ©rÃ¼lt-e
- Viszony rÃ©teg
    - kÃ©t szÃ¡mÃtÃ³gÃ©p felhasznÃ¡lÃ³i kapcsolatot lÃ©tesÃtsen
    - Ã¡llomÃ¡nyokat mozgathatunk
- MegjelenÃtÃ©si rÃ©teg
    - Ã¡tvitt informÃ¡ciÃ³ szintaktikÃ¡ja, szemantikÃ¡ja
    - a pÃ¡rbeszÃ©d sorÃ¡n absztrakt mÃ³don kell definiÃ¡lni a kÃ³dolÃ¡sokat
- AlkalmazÃ¡si rÃ©teg
    - protokollok sokasÃ¡ga, HTTP, FTP

Internet

Ãsszekapcsolt szÃ¡mÃtÃ³gÃ©pes hÃ¡lÃ³zatok globÃ¡lis rendszere, ami a TCP/IP protokollt hasznÃ¡lja a kommunikÃ¡ciÃ³hoz. Olyan hÃ¡lÃ³zatok hÃ¡lÃ³zata, amely Ã¼zleti, kormÃ¡nyzati, Ã¡llai, magÃ¡n, tudomÃ¡nyos stb hÃ¡lÃ³zatokbÃ³l Ã¡ll. KÃ¶zÃ¶s protokollokat hasznÃ¡lnak Ã©s kÃ¶zÃ¶s szolgÃ¡ltatÃ¡sokat nyÃºjtanak.

Nincs kÃ¶zpontosÃtott irÃ¡nyÃtÃ¡sa, sem a technolÃ³giai megvalÃ³sÃtÃ¡sban, sem a hozzÃ¡fÃ©rÃ©sre Ã©s hasznÃ¡latra vonatkozÃ³ politikÃ¡ban.

ElsÅdleges elÅfutÃ¡r-hÃ¡lÃ³zata az ARPANET volt, ami regionÃ¡lis tudomÃ¡nyos Ã©s katonai hÃ¡lÃ³zatok Ã¶sszekapcsolÃ¡sÃ¡nak gerincÃ©t szolgÃ¡ltatta. MiutÃ¡n a TCP/IP lett az egyetlen hivatalos protokoll rajta, gyorsan nÅtt a hozzÃ¡ csatlakozÃ³ hÃ¡lÃ³zatok, gÃ©pek Ã©s felhasznÃ¡lÃ³k szÃ¡ma.

AzÃ³ta mÃ©gtÃ¶bb terÃ¼let csatlakozott hozzÃ¡, globÃ¡lis gerinchÃ¡lÃ³zatok Ã©pÃ¼ltek ki.

Egy gÃ©p rajta van az interneten, ha a TCP/IP protokollt hasznÃ¡lja, van sajÃ¡t IP-je, Ã©s tud mÃ¡s gÃ©peknek csomagokat kÃ¼ldeni az interneten Ã¡t.

FÅ alkalmazÃ¡si terÃ¼letek hagyomÃ¡nyosan:

- e-levÃ©l
- hÃrek
- tÃ¡voli bejelentkezÃ©s
- fÃ¡jltranszfer

Egy Ãºj alkalmazÃ¡s, a WWW bevezetÃ©se vont be tÃ¶bb milliÃ³ Ãºj felhasznÃ¡lÃ³t a hÃ¡lÃ³zatba. Nem vÃ¡ltozatott semmit az rendelkezÃ©sre Ã¡llÃ³ eszkÃ¶zÃ¶kÃ¶n, csak egyszerÅ±bbÃ© tette a hasznÃ¡latukat. A bÃ¶ngÃ©szÅk megjelenÃ©sÃ©vel kÃ©peket, szÃ¶veget tartalmazÃ³ oldalakra is el lehetett jutni, Ã©s onnan mÃ¡s oldalakra tovÃ¡bbnavigÃ¡lni.

A nÃ¶vekedÃ©s nagy rÃ©sze az Ãºn. ISP-knek is kÃ¶szÃ¶nhetÅ. EgyÃ©ni felhasznÃ¡lÃ³knak nyÃºjtanak szolgÃ¡ltatÃ¡sokat, internetelÃ©rÃ©st.

ITU

International Telecommunication Union - NemzetkÃ¶zi TÃ¡vkÃ¶zlÃ©si egyesÃ¼let

SzÃ¼ksÃ©g van vilÃ¡gmÃ©retÅ± kompatibilitÃ¡sra, hogy a kÃ¼lÃ¶nbÃ¶zÅ orszÃ¡gokban Ã©lÅ emberek/szÃ¡mÃtÃ³gÃ©pek kapcsolatba kerÃ¼lhessenek egymÃ¡ssal. A feladata az, hogy szabvÃ¡nyosÃtsa a nemzetkÃ¶zi tÃ¡vkÃ¶zlÃ©st.

HÃ¡rom fÅ Ã¡gazata van:

- ITU-R: rÃ¡diÃ³kommunikÃ¡ciÃ³s Ã¡gazat
- ITU-T: tÃ¡vkÃ¶zlÃ©si szabvÃ¡nyosÃtÃ¡si Ã¡gazat
- ITU-D: fejlesztÃ©si Ã¡gazat

ITU-R

Az 1927-ben NemzetkÃ¶zi RÃ¡diÃ³ TanÃ¡csadÃ³ BizottsÃ¡g vagy CCIR nÃ©ven (francia nevÃ©n ComitÃ© consultatif international pour la radio) alapÃtott Ã¡gazat kezeli a nemzetkÃ¶zi rÃ¡diÃ³frekvenciÃ¡s spektrum- Ã©s mÅ±holdpÃ¡lya-erÅforrÃ¡sokat. 1992-ben a CCIR lett az

ITU-R. Feladata a raÌdioÌfrekvenciaÌk kiosztaÌsa a vilaÌgszerte egymaÌssal versengoÌ csoportoknak.

ITU-T

A szabvÃ¡nyosÃtÃ¡s a kezdetektÅl fogva cÃ©lja az ITU-nak. 1956-ban a NemzetkÃ¶zi Telefon- Ã©s TÃ¡virati TanÃ¡csadÃ³ BizottsÃ¡g egysÃ©gesÃti a globÃ¡lis tÃ¡vkÃ¶zlÃ©st.

Az ITU-T feladata, hogy mÅ±szaki javaslatokat tegyen az adatkommunikÃ¡ciÃ³ interfÃ©szeire. Ezek gyakran vÃ¡lnak nemzetkÃ¶zi szabvÃ¡nyokkÃ¡. Fontos, hogy ezek csak mÅ±szaki javaslatokat tartalmaznak. Az elfogadÃ¡sa csak az adott orszÃ¡gon mÃºlik.

ITU-D

Az 1992-ben lÃ©trehozott Ã¡gazat hozzÃ¡jÃ¡rul az informÃ¡ciÃ³s Ã©s kommunikÃ¡ciÃ³s technolÃ³giÃ¡khoz (IKT) valÃ³ igazsÃ¡gos, fenntarthatÃ³ Ã©s megfizethetÅ hozzÃ¡fÃ©rÃ©s terjesztÃ©sÃ©hez.

IEEE

Villamos Ã©s Elektronikai MÃ©rnÃ¶kÃ¶k IntÃ©zete

A vilÃ¡g legnagyobb szakmai szervezete. KonferenciÃ¡k Ã©s folyÃ³iratok mellett szabvÃ¡nyokat dolgoznak ki a villamosmÃ©rnÃ¶ki tudomÃ¡nyok Ã©s az informatika terÃ©n.

Az IEEE 802-es bizottsÃ¡ga tÃ¶bb LAN fajtÃ¡t szabvÃ¡nyosÃtott. A sikertÃ¶rtÃ©netek (802.3 Ã©s 802.11, logikai kapcsolatvezÃ©rlÃ©s Ã©s vezetÃ©k nÃ©lkÃ¼li LAN) hatÃ¡sa Ã³riÃ¡si volt.# 14. Kiemelt fontossÃ¡gÃº kommunikÃ¡ciÃ³s protokollok (PPP, Ethernet, IP, TCP, HTTP, RSA)

PPP

Magas szintÅ± adatkapcsolati protokoll kÃ©tpontos vonalakhoz. MindenfÃ©le fizikai rÃ©tegek feletti hasznÃ¡latra alkalmas.

SzolgÃ¡ltatÃ¡sai:

- egyÃ©rtelmÅ±en Ã¡brÃ¡zolja a keret vÃ©gÃ©t Ã©s a kÃ¶vetkezÅ keret elejÃ©t, a keretformÃ¡tum megoldja a hibajelzÃ©st is
- adatkapcsolat-vezÃ©rlÅ protokollt tartalmaz a vonalak felÃ©lesztÃ©sÃ©re, tesztelÃ©sÃ©re, vonalak bontÃ¡sÃ¡ra
- kÃ¼lÃ¶nbÃ¶zÅ hÃ¡lÃ³zati vezÃ©rlÅ protokollokat tartalmaz mindegyik tÃ¡mogatott hÃ¡lÃ³zati rÃ©teghez

Ethernet

Az Ethernet egy szÃ¡mÃtÃ³gÃ©pes hÃ¡lÃ³zati technolÃ³giÃ¡k csalÃ¡dja, amelyet helyi hÃ¡lozatban (LAN), vÃ¡rosi hÃ¡lÃ³zatokban (MAN) Ã©s nagy kiterjedÃ©sÅ± hÃ¡lÃ³zatokban (WAN) hasznÃ¡lnak. ElÅszÃ¶r 1983-ban szabvÃ¡nyosÃtottÃ¡k IEEE 802.3 nÃ©ven. Az Ethernet-et azÃ³ta finomÃtottÃ¡k, hogy tÃ¡mogassa a nagyobb bitsebessÃ©get, a nagyobb csomÃ³pontok szÃ¡mÃ¡t Ã©s a nagyobb Ã¶sszekÃ¶ttetÃ©si tÃ¡volsÃ¡gokat.

Az Ethernet egy Ã¡llomÃ¡sa a kÃ¶zvetÃtÅ kÃ¶zeggel (kÃ¡bel) valÃ³ Ã¡llandÃ³ kapcsolatot kihasznÃ¡lva bele tud hallgatni a csatornÃ¡ba, Ãgy ki tudja vÃ¡rni, amÃg a csatorna felszabadul, Ã©s a sajÃ¡t Ã¼zenetÃ©t leadhatja anÃ©lkÃ¼l, hogy ezzel mÃ¡s Ã¼zenet sÃ©rÃ¼ljÃ¶n, tehÃ¡t a torlÃ³dÃ¡s elkerÃ¼lhetÅ. A csatornÃ¡t az Ã¡llomÃ¡sok folyamatosan figyelik, ha Ã¼tkÃ¶zÃ©st tapasztalnak, akkor zavarni kezdik a csatornÃ¡t, hogy figyelmeztessÃ©k a kÃ¼ldÅket, ezutÃ¡n vÃ©letlen ideig vÃ¡rnak, majd adni kezdenek. Ha ezek utÃ¡n tovÃ¡bbi Ã¼tkÃ¶zÃ©sek tÃ¶rtÃ©nnek, az eljÃ¡rÃ¡s ugyanez, de a vÃ©letlenszerÅ± vÃ¡rakozÃ¡s idejÃ©t kÃ©tszeresÃ©re nÃ¶velik, Ãgy idÅben szÃ©tszÃ³rjÃ¡k a versenyhelyzeteket, esÃ©lyt adva arra, hogy valaki adni tudjon.

IP

Az internet hÃ¡lÃ³zat egyik alapvetÅ szabvÃ¡nya (avagy protokollja). Ezen protokoll segÃtsÃ©gÃ©vel kommunikÃ¡lnak egymÃ¡ssal az internetre kÃ¶tÃ¶tt csomÃ³pontok (szÃ¡mÃtÃ³gÃ©pek, hÃ¡lÃ³zati eszkÃ¶zÃ¶k, webkamerÃ¡k stb.). A protokoll meghatÃ¡rozza az egymÃ¡snak kÃ¼ldhetÅ Ã¼zenetek felÃ©pÃtÃ©sÃ©t, sorrendjÃ©t stb.

A k

TCP

HTTP

RSA