1 ALGORITMY A ALGORITMIZÁCIA ÚLOH

1.1 Etapy algoritmizácie úloh

Spracovanie informácií predstavuje proces, v ktorom sú konkrétne vstupné údaje pretvárané do výsledkov, ktoré možno použiť na riadenie a rozhodovanie. Ak chceme riešiť akúkoľvek úlohu na počítači, treba ju rozdeliť na celý rad prípravných prác – etáp.

Algoritmizácia úloh má tri základné etapy:
- formulácia úlohy,
- analýza úlohy,
- zostavenie riešiaceho algoritmu.

Formulácia úlohy

Prvým predpokladom, aby sme danú úlohu mohli riešiť na počítači, je jej jasná a jednoznačná formulácia a identifikácia, ako aj ujasnenie cieľa, ktorý sledujeme riešením príslušnej úlohy. Za tým nasleduje tzv. formulácia problému, napr. matematickými prostriedkami (modelom), čiže problém musíme formalizovať pomocou nejakej sústavy vzťahov medzi premennými a konštantami. Formalizovanie konkrétnej úlohy si spravidla vyžaduje individuálny prístup, adaptáciu štandardných postupov, príp. nový typ modelu. Na formalizáciu možno použiť aj iný spôsob ako matematický, môže to byť napr. grafický model. Pre číslicové počítače je však matematická formulácia najvhodnejšia.

Analýza úlohy

V tejto etape je potrebné nájsť algoritmus riešenia úlohy. Zisťuje sa, či úloha je riešiteľná, či má jedno alebo viac riešení, načrtávajú sa možnosti riešenia a rozhoduje sa o druhu metód. Vytypovaná metóda riešenia musí zabezpečovať dosiahnutie požadovaných výsledkov (výstupné informácie), ale zároveň musí presne určiť, ktoré vstupné údaje budú potrebné. Úloha sa zovšeobecňuje a uskutočňuje sa prvá predstava o algoritmickej riešiteľnosti.

Zostavenie riešiaceho algoritmu

Po správnej formulácii a analýze úlohy nasleduje etapa syntetickej činnosti, v ktorej sa popíše logika a postup riešenia úlohy. Výsledkom tejto etapy je riešiaci algoritmus.

Do tejto etapy môžeme zahrnúť aj *programovanie úlohy*. Pod pojmom programovanie rozumieme činnosť, pomocou ktorej sa uskutočňuje prevod úlohy z ľudského vedomia do formy vhodnej pre spracovanie na počítači. Výsledkom tejto činnosti je program. Program je algoritmus v takej forme, ktorej rozumie počítač, t.j. program je zápis algoritmu v niektorom programovacom jazyku.

1.2 Algoritmus

Predpokladom riešenia úloh na počítači je existencia algoritmu. Pod pojmom *algoritmus* rozumieme presný a jednoznačný predpis postupu riešenia úlohy, t. j. proces transformácie vstupných údajov na požadované výsledky. Tento predpis sa skladá z jednotlivých krokov (elementárnych operácií), ktoré sa vykonávajú v určenom poradí. Počet výpočtových krokov musí byť konečný.

Algoritmus nemusí popisovať len výpočtový proces, môže popisovať aj geometrickú konštrukciu, alebo aj inú činnosť (denný režim, recept na prípravu jedla a pod.).

Algoritmus má 3 základné vlastnosti:
- determinovanosť,
- rezultatívnosť,
- hromadnosť.

Determinovanosť

Algoritmus determinuje (t. j. presne určuje) proces pretvárania vstupných údajov na výsledky. V každom kroku musí byť presne určené, ktorý krok sa vykoná ako ďalší, alebo či algoritmus končí. To znamená, že výpočet sa môže uskutočniť podľa vopred daných pokynov (predpisov) o operáciách, pričom toto uskutočnenie nezávisí od osoby, ktorá ich vykonáva. Postup od vstupných údajov až k výsledným je presne určený a vykonávateľ algoritmu vykonáva, možno povedať, čisto mechanickú prácu bez náhodných vplyvov.

Rezultatívnosť

Táto vlastnosť znamená, že algoritmus použitý na ľubovoľnú úlohu z tej triedy úloh, pre ktorú bol zostavený, zabezpečí výsledok po konečnom počte krokov, t. j. algoritmus vedie k výsledku pre všetky prípustné vstupné údaje. Mimo tejto oblasti by algoritmus nemusel viesť k správnym výsledkom. Pre ľubovoľnú n-ticu vstupných údajov z určitej množiny M vedie algoritmus vždy k hľadanému výsledku. Množinu M nazývame oblasťou použiteľnosti daného algoritmu.

Hromadnosť

Algoritmus musí byť zostavený tak, aby riešil veľkú, obyčajne nekonečnú triedu úloh rovnakého typu. Musí to byť popis riešenia nie jednej konkrétnej úlohy, ale celej skupiny príbuzných úloh, ktoré sa odlišujú len hodnotami vstupných údajov.

Postup riešenia úlohy, ktorý nevyhovuje každej z uvedených troch vlastností, nemožno považovať za algoritmus.

✍ *Pojem algoritmu vznikol v súvislosti s hľadaním spôsobov, ako riešiť všetky úlohy danej skupiny. Najznámejšie sú algoritmy, ktoré riešia numerické úlohy. Patria medzi ne pravidlá, pomocou ktorých sa vykonávajú základné počtové úkony v desiatkovej číselnej sústave. Tie algoritmy, v ktorých základnú úlohu plnia štyri základné počtové úkony a v ktorých sú vstupnými i výstupnými údajmi čísla, nazývame číselnými algoritmami. Sú veľmi rozšírené, čo sa vysvetľuje možnosťou redukovania iných operácií na tieto štyri základné.*

✍ *Skupina úloh určitého druhu sa považuje za riešiteľnú, ak je pre jej vyriešenie vytvorený algoritmus. Cieľom matematiky je hľadať takéto algoritmy. Ak nemožno nájsť algoritmus na riešenie všetkých úloh danej skupiny, potom treba nájsť špeciálny postup riešenia pre každý jednotlivý prípad.*

Algoritmus možno vyjadriť rôznymi prostriedkami: slovným popisom, matematickým jazykom (rovnicami, maticami, vzorcami a pod.), alebo iným spôsobom. Závisí od toho, komu a na aký účel je daný algoritmus určený. Riešenie algoritmu na počítači nemusí byť vždy riešením číselného algoritmu, ktoré spočíva v postupnosti základných aritmetických operácií v príslušnej číselnej sústave. Nečíselné algoritmy majú tiež veľký podiel vo využívaní výpočtovej techniky. Nečíselné algoritmy, ako sú

rozhodovacie procesy v riadiacich systémoch, počítačové riešenie hier, hľadanie v zoznamoch, strojový preklad textu a iné, spočívajú spravidla na logických operáciách počítača. V bežnom dennom živote existuje nekonečne veľa algoritmov, ktoré používame, bez toho, aby sme si to uvedomovali (napr. návod na obsluhu stroja, technologické postupy vo výrobe, a pod.).

Úlohy z hľadiska algoritmizovateľnosti rozlišujeme:

- *Úlohy ľahko algoritmizovateľné* – na ich riešenie použijeme niektorý známy algoritmus alebo bez väčších problémov zostavíme algoritmus sami.
- Úlohy, ktoré sú považované za veľmi zložité a *ako celok nealgoritmizovateľné.* Ak to je možné, rozložíme úlohy na niekoľko dielčích častí, pre ktoré už môžeme algoritmus zostaviť.
- *Úlohy nealgoritmizovateľné* – úlohy, ktoré nemajú jednoznačne popísaný postup a u ktorých je v ľubovoľnom okamžiku možné akékoľvek riešenie. V skutočnosti nealgoritmizovateľných úloh je veľmi málo a veľakrát sa za nealgoritmizovateľné považovali úlohy, ktoré boli nedostatočne formulované.

V mnohých prípadoch sa stáva, že myšlienka vytvorenia algoritmu je z teoretického hľadiska veľmi jednoduchá, ale pri praktickej realizácii sa vyskytujú ťažkosti (napr. šach, matematické rébusy). Tieto algoritmy sú z praktického hľadiska veľmi rozsiahle, pozostávajú z veľkého počtu operácií. Pri riešení úloh takéhoto druhu sa používa *heuristický* spôsob hľadania riešenia. Táto metóda však nezaručuje nájdenie presného, resp. optimálneho riešenia a nemusí zaručiť ani nájdenie riešenia vôbec. V poslednom období heuristická metóda prichádza do popredia najmä v súvislosti s používaním počítačov na riešenie nenumerických úloh.

1.3 Vývojové diagramy

Algoritmus môže byť vyjadrený rôznym spôsobom, verbálne, písomne a pod. Vo výpočtovej technike sa používa grafická forma zápisu algoritmu pomocou tzv. *vývojového diagramu (VD).*

Vývojový diagram je blokový diagram určitého procesu (napr. programu), vyjadruje jeho štruktúru a nadväznosť činností.

Vývojový diagram programu graficky znázorňuje logickú stavbu programu pre systém spracovania informácií.

✍ *Na kreslenie vývojových diagramov sa používajú grafické značky, ktoré umožňujú dobrú vizuálnu orientáciu v štruktúre aj zložitých algoritmov. Značky používané vo vývojových diagramoch sú definované a popísané v STN 36 9030. Z nich vyberieme súbor značiek najviac používaných vo vývojových diagramoch programov.*

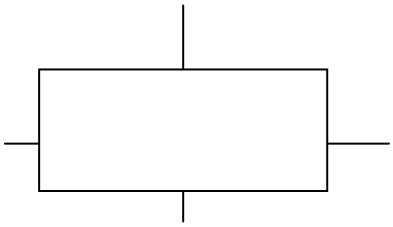
1.3.1 Značky vývojových diagramov

Značky vývojových diagramov sa kreslia podľa šablóny *LOGAREX 25518.* V súčasnosti existujú rôzne druhy softvéru na kreslenie vývojových diagramov. Na kreslenie VD v tejto učebnej pomôcke je použitý programový produkt *Visio.*

Do značiek sa zapisujú slovne alebo symbolicky aritmetické alebo logické operácie (skupiny operácií) alebo symboly bližšie určujúce účel značky. Značky sa spájajú spojnicami. Vstup do značky zhora alebo zľava, ako aj výstup zo značky dolu alebo

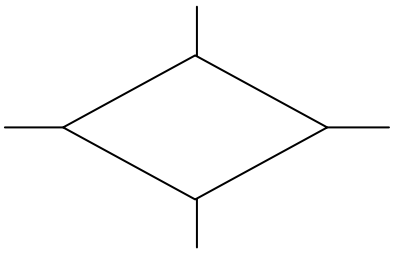
doprava, netreba označovať šípkou (tzv. preferovaný smer). Ostatné smery vstupov a výstupov sa označujú *šípkou* nepreferovaný smer).

Rozmery značiek nie sú predpísané, záväzný je iba pomer strán a vnútorné uhly. Veľkosť značky volíme tak, aby sme mohli do nej prehľadne umiestniť potrebné slová alebo symboly.



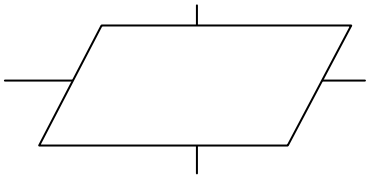
Spracovanie

Vykonanie akejkoľvek operácie, ktorej výsledkom je transformovanie informácie.



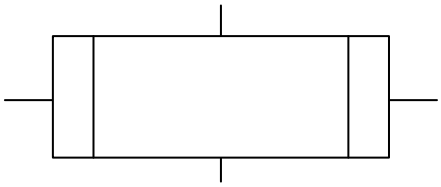
Rozhodovanie, vetvenie, prepínanie

Predstavuje rozhodovací alebo prepínací typ operácie, ktorá určuje alternatívnu cestu ďalšieho vývoja diagramu.



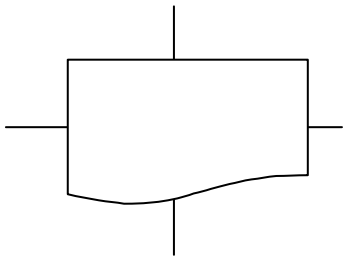
Vstup – výstup

Vstupná alebo výstupná činnosť, t. j. dodanie informácie pre spracovanie (vstup) alebo záznam spracovanej informácie (výstup).



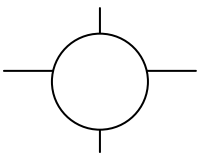
Predom definovaná činnosť

Skupina činností alebo krokov, ktoré sú špecifikované na inom mieste a v danom vývojovom diagrame nie sú rozpracované.



Doklad

Vstupná alebo výstupná operácia, v ktorej je médium doklad, napr. výpis zostavy na tlačiarni.



Spojka

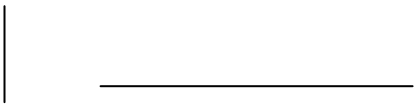
Prechod na inú časť alebo z inej časti vývojového diagramu.

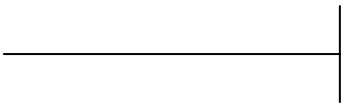
Medzná značka

Označenie medzného bodu (miesta) vývojového diagramu, napr. začiatku, konca, zastavenia.

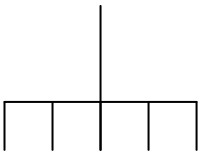
Spojnice

Slúžia na spojenie jednotlivých značiek.





Spojenie spojníc do jednej výslednej.



Vetvenie spojníc (jedna spojnica sa vetví na niekoľko spojníc).

1.3.2 Druhy vývojových diagramov

Vývojové diagramy sa môžu používať v rôznych situáciách. Môžu znázorňovať len rámcový projekt spracovania údajov, alebo môžu podrobne znázorňovať všetky logické a výpočtové operácie. V každom prípade je zostavený vývojový diagram nástrojom analýzy, a preto aj úroveň vývojového diagramu musí byť podriadená potrebám analýzy. Medzi najhrubšími vývojovými diagramami (v ktorých jednou značkou sa znázorní jeden celý program) a najpodrobnejšími vývojovými diagramami (v ktorých jedna značka znázorňuje jednu operáciu) je niekoľko stupňov podrobností.

Podľa toho možno vývojové diagramy rozdeliť do nasledujúcich skupín:

- hrubé vývojové diagramy,
- analytické vývojové diagramy,
- vývojové diagramy programu.

Hrubý vývojový diagram predstavuje vývojový diagram úlohy (projektu). Popisuje hlavné kroky algoritmu riešenej úlohy a používa sa prevažne vtedy, ak je úloha veľmi rozsiahla a delí sa na viac programov, ktoré sa predpokladajú postupne spracovávať na počítači. Takáto úloha a jej spracovanie sa skladá z väčšieho počtu úsekov. Vývojový diagram tohto druhu sa označuje i ako *ideový* vývojový diagram, pri jeho kreslení sa používajú niektoré formálne výnimky. Na rozdiel od normálneho zaznamenávania obdĺžnika v hrubom vývojovom diagrame, sa kreslí niekedy na výšku, zaznamenáva sa postup údajov aj v priebehu riešenia a používa sa toľko spojníc pre vstupy, koľko je použitých vstupných kanálov a toľko výstupných spojníc, koľko je použitých

výstupných kanálov. Postup údajov v priebehu riešenia je zapísaný tak z jedného nositeľa informácií k druhému, ako aj postup od jednej čiastkovej úlohy k druhej. Dôraz sa teda kladie nielen na vlastný výpočet, ale aj na pohyb, tok informácií v priebehu celkového riešenia. Jednotlivé značky vývojového diagramu sú pomenované podľa úseku, ktorý znázorňujú.

V hrubom vývojovom diagrame sa často rieši aj niekoľko na seba nadväzujúcich čiastkových úloh. Existuje tu viac vstupných informácií na rôznych nosičoch informácií, v jednom celkovom riešení existuje rôzny časový rytmus jeho jednotlivých úsekov a pod. Aj tento druh vývojových diagramov je nástrojom analýzy a predovšetkým musí preto vyhovovať jej potrebám aj proti prísnym formálnym pravidlám.

Hrubé vývojové diagramy majú tieto vlastnosti:
- neriešia sa na tomto stupni špecifické obraty,
- popisujú sa len hlavné kroky algoritmu riešenia,
- neuvažuje sa so špeciálnymi vlastnosťami počítača.

Jednotlivé úseky hrubého vývojového diagramu podrobnejšie rozpisujeme v analytických vývojových diagramoch.

Analytické vývojové diagramy znázorňujú podrobnejšie logické a výpočtové operácie, ale nie sú ešte postavené pre určitý typ počítača. Analytické vývojové diagramy vypĺňajú celú škálu vývojových diagramov medzi hrubými a programovými vývojovými diagramami. Táto hierarchia vývojových diagramov poskytuje pri riešení úloh neoceniteľné výhody. Dovoľuje postupovať od riešenia základnej logickej štruktúry v hrubých rysoch k detailom. Preto sa často v literatúre stretávame s delením analytických vývojových diagramov na *hrubé analytické* a *podrobné analytické* diagramy. Rozdiel jednotlivých druhov vyplýva z pomenovania.

Analytické vývojové diagramy okrem toho, že slúžia pre kontrolu a rozpis hrubých vývojových diagramov, umožňujú preverovať správnosť a reálnosť ideového projektu a sprísniť nároky na niektoré technické nároky počítača. Zároveň umožňujú preveriť a spresniť nadväznosť hlavných úsekov, potrebu jednotlivých úsekov diagramu a ich rozsah. Správnosť logických nadväzností a postupov príslušného riešenia čiastkovej úlohy v analytickom vývojovom diagrame sa preveruje simuláciou.

Vývojové diagramy programu sú najpodrobnejšie vývojové diagramy. Je to vlastne rozpracovanie čiastkových alebo osobitne zložitých analytických diagramov.

Vývojové diagramy programu majú tieto typické vlastnosti:
- vývojový diagram je už prispôsobený na určitý technický typ počítača, jeho vlastnostiam a obsahu jeho operácie,
- ak počítač používa symbolické programovanie, tak vývojový diagram môže obsahovať tieto symboly,
- jednou značkou sa spravidla vždy znázorňuje jedna operácia,
- má úzku nadväznosť na programovací jazyk.

Vývojové diagramy programu sú hlavným podkladom pre zostavenie cieľového programu určitého počítača pri použití určitého programovacieho jazyka. Často je výhodné zostaviť niekoľko vývojových diagramov programu a z nich potom vybrať ten, ktorý najlepšie zlaďuje požiadavky úlohy s možnosťami konkrétneho počítača.

Vývojové diagramy programu podľa postupu, ako jednotlivé operácie na seba nadväzujú, možno rozdeliť na:

- vývojové diagramy priame,
- vývojové diagramy s vetvením.

Pri *vývojových diagramoch priamych* v postupe neprichádza k žiadnemu alternatívnemu postupu. Znázorňujú proces, ktorý je priamočiary.

Vývojové diagramy vetvené možno rozdeliť na:

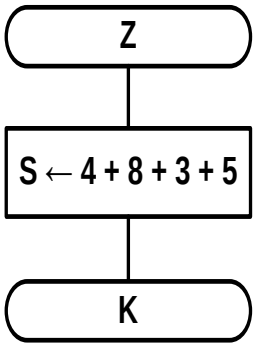
- vývojové diagramy vetvené bez opakovania,
- vývojové diagramy vetvené s opakovaním (s cyklom).

Vývojové diagramy vetvené bez opakovania znázorňujú proces, v ktorom prichádza k altenatívnemu postupu (minimálne dve možnosti), ale ani jedna časť sa neopakuje.

Pri vývojových diagramoch vetvených s opakovaním prichádza taktiež k alternatívnemu riešeniu (rozhodovaniu) a jedna z častí procesu znázorňovaná vývojovým diagramom, sa niekoľkokrát opakuje za zmenených podmienok. Tejto časti sa hovorí *cyklus*.

Vývojový diagram programu priamy

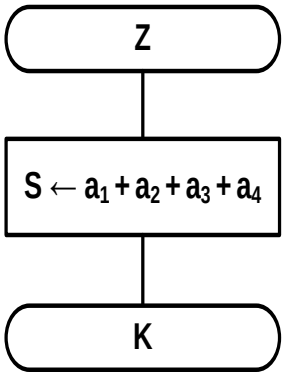
Jeho znázornenie je najvhodnejšie na praktickom príklade. Máme spočítať 4 celé kladné čísla 4, 8, 3, 5. Vývojový diagram bude nasledovný:



Obr. 3.1 VD priamy

Vývojový diagram bude však vhodnejšie zostaviť s obecným označením sčítancov. V takom prípade bude jeho platnosť pre ľubovoľnú štvoricu sčítancov dosadenú za obecné označenie. Pre obecné označenie môžeme voliť označenie abecednými znakmi, v našom prípade napr.: a, b, c, d, alebo označenie abecedným znakom s indexami, v našom prípade napr.: a_1, a_2, a_3, a_4. Platí teda: $a_1 = 4$, $a_2 = 8$, $a_3 = 3$, $a_4 = 5$.

Vývojový diagram bude potom:

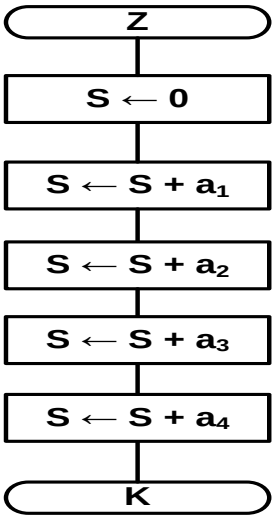


Obr. 3.2 VD s obecným označením sčítancov

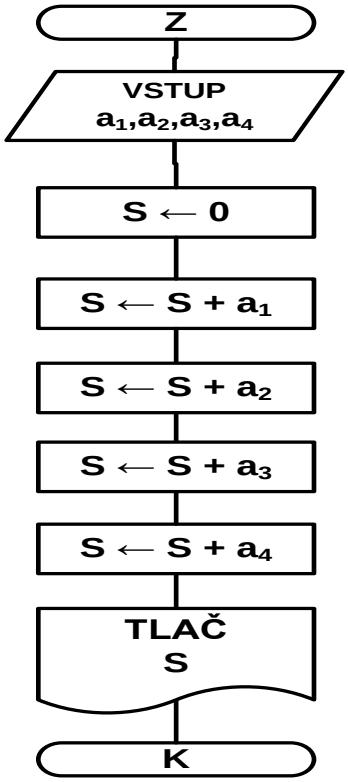
Vo vývojovom diagrame možno úlohu rozložiť do jednoduchých operácií na elementárne kroky. Každé sčítanie potom treba posudzovať ako samostatnú operáciu a pri riešení sa bude postupovať takto:

- označíme súčet sčítancov (v našom prípade S),
- rozložíme úlohu na elementárne kroky tak, aby pre každú operáciu sčítania bola jedna značka,
- upravíme, aby boli operácie rovnorodé tak, že súčet sčítancov položíme rovný nule.

Vývojový diagram bude potom takýto:



Obr. 3.3



Obr. 3.4

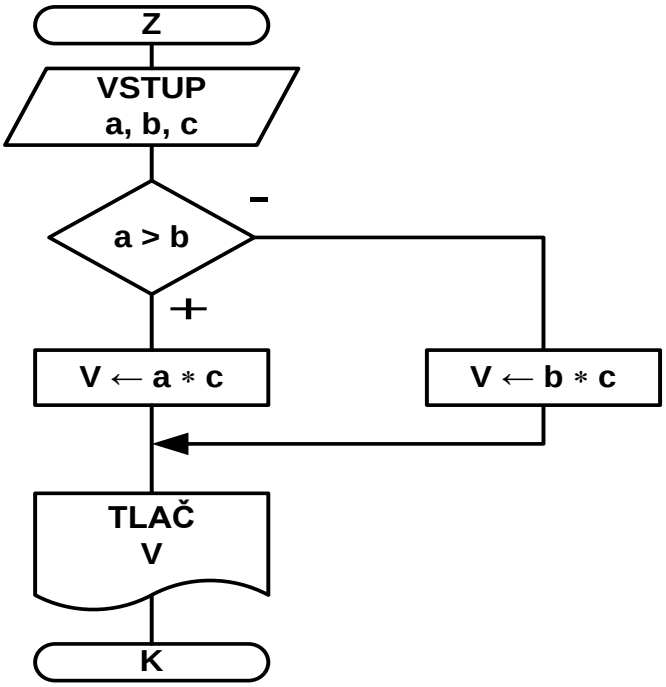
V prípade, že spracovávané sčítance sú zadávané na vstupnom prostriedku, je potrebné vo vývojovom diagrame znázorniť operáciu vstupu. Taktiež, ak chceme znázorniť výsledok (napr. vytlačiť), pribudne operácia výstupu. V tom prípade je vývojový diagram zobrazený na obr. 3.4.

Vývojový diagram programu s vetvením bez opakovania

Na rozdiel od predchádzajúceho typu vývojového diagramu, vývojové diagramy s vetvením obsahujú rozhodovacie značky znázorňujúce rozhodovacie operácie, ktoré diagram vetvia, určujú alternatívne postupy.

Riešenie takéhoto prípadu si ukážeme opäť na konkrétnom príklade:

Sú zadané tri čísla (predpokladajme, že sú rôzne, t.j. a ≠ b ≠ c). Z prvých dvoch treba nájsť väčšie, vynásobiť ho tretím a výsledok vytlačiť. Z obsahu príkladu je zrejmé, že je potrebné porovnávaním čísel nájsť číslo väčšie. Vlastne sa rozhoduje medzi dvoma možnosťami, teda použije sa rozhodovacia značka s otázkou. Odpovede na danú otázku vyjadrujú alternatívne možnosti ďalšieho postupu. Nájdené väčšie číslo vynásobíme tretím a získaný výsledok vytlačíme. Keďže označenie čísel je **a, b, c** a výsledok **V**, vývojový diagram tohto príkladu bude nasledovný:



Obr. 3.1 VD s vetvením bez opakovania

Vetvenie v tomto VD je *dvojité (dichotomické)*, t.j. značka pre rozhodovanie má dva výstupy. Výsledok testovania je *áno* (označenie +) alebo *nie* (označenie –).

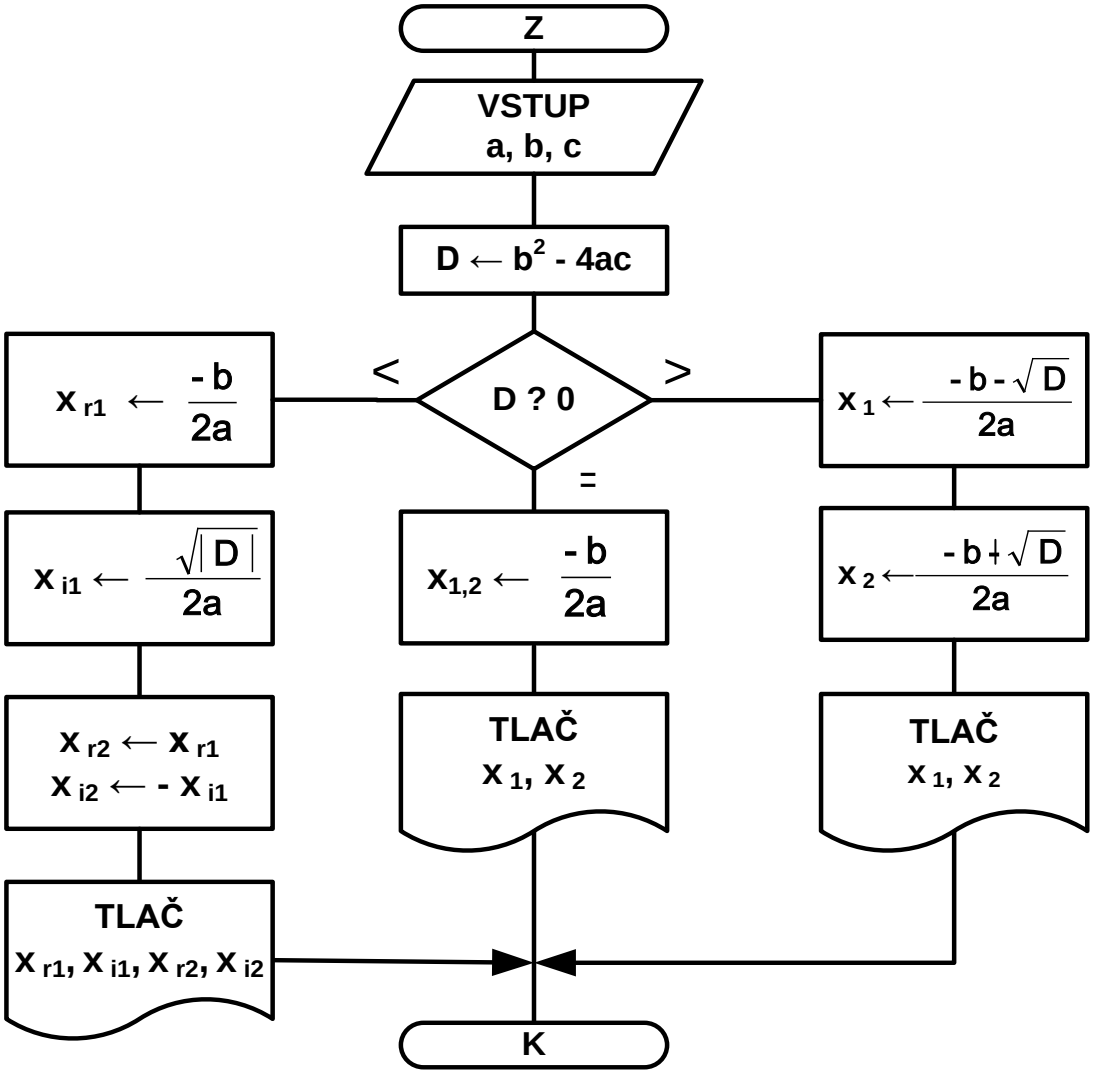
Konkrétne v tomto príklade, ak predpokladáme, že a ≠ b ≠ c, pri porovnávaní prvých dvoch čísel sú len dve možnosti, a>b alebo b>a.

Vetvenie vo vývojových diagramoch môže byť aj *trojité (trichotomické)*, t.j. značka pre rozhodovanie má tri výstupy.

Príkladom na trojité vetvenie je výpočet koreňov kvadratickej rovnice $\mathbf{ax^2 + bx + c = 0}$, kde testujeme hodnotu vypočítaného diskriminantu D.

Diskriminant, ktorý vypočítame ako $\mathbf{D=b^2–4ac}$, môže byť D>0, D=0 alebo D<0.

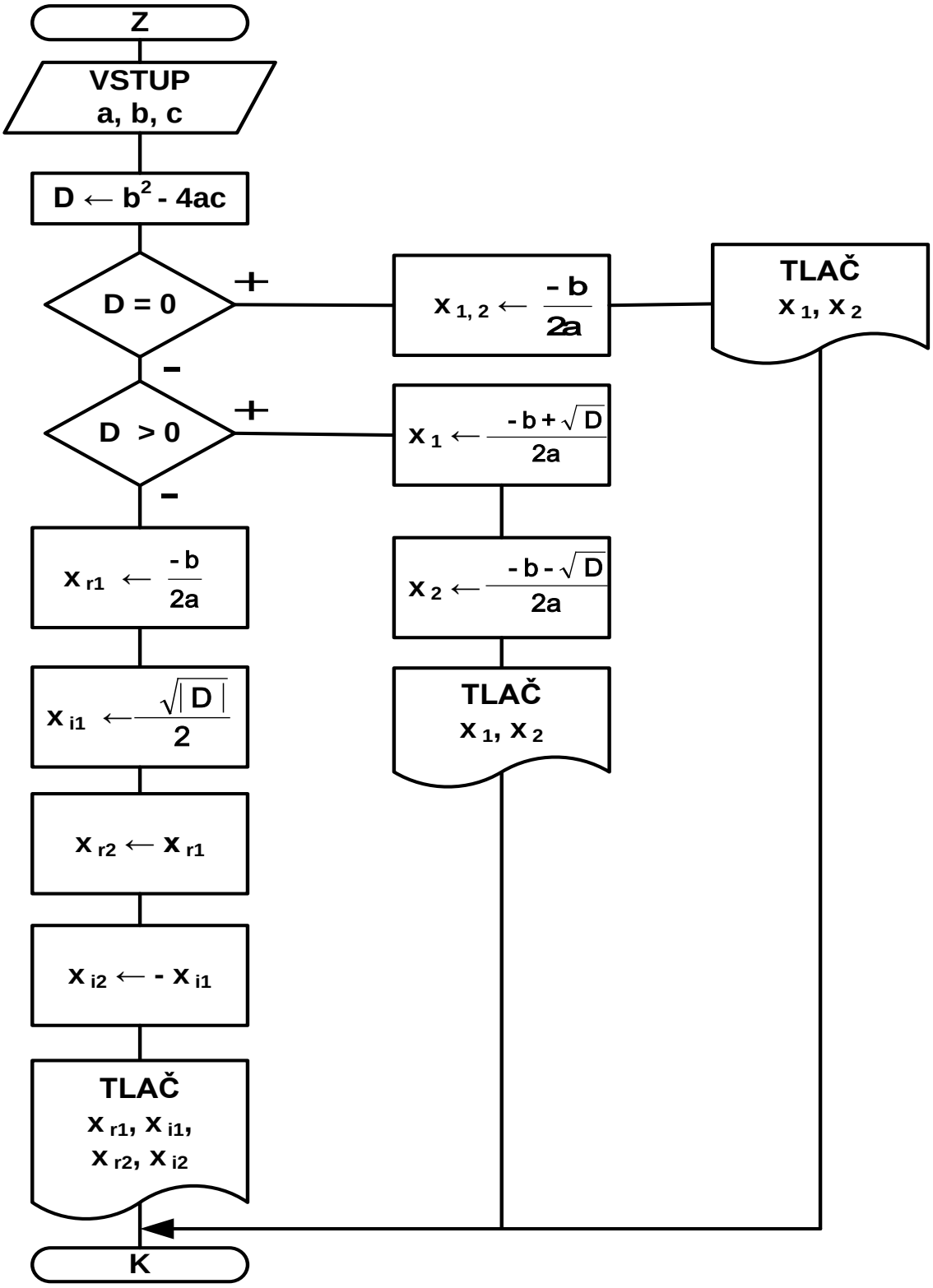
Ak použijeme *trojité* (trichotomické) vetvenie, vývojový diagram na výpočet koreňov kvadratickej rovnice je potom takýto:



Obr. 3.2 a) VD – výpočet koreňov kvadratickej rovnice

Otázka v rozhodovaní na hodnotu diskriminantu sa môže položiť i inak a vývojový diagram pre daný príklad bude mať dve rozhodovacie značky, každú s dvoma výstupmi: (obr. 3.7)

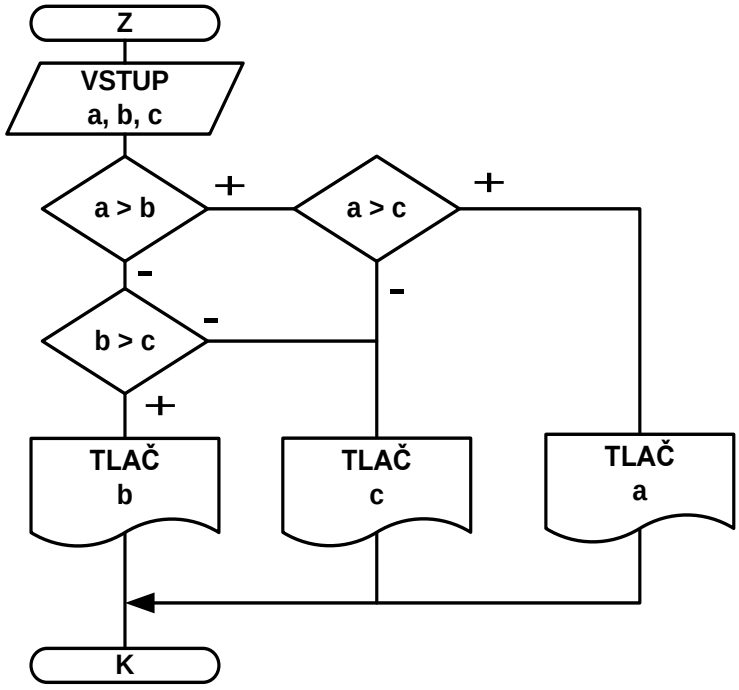
✍ *Každé trojité (trichotomické) vetvenie vo vývojovom diagrame sa dá zapísať aj ako postupnosť dvoch dvojitých (dichotomických) vetvení.*

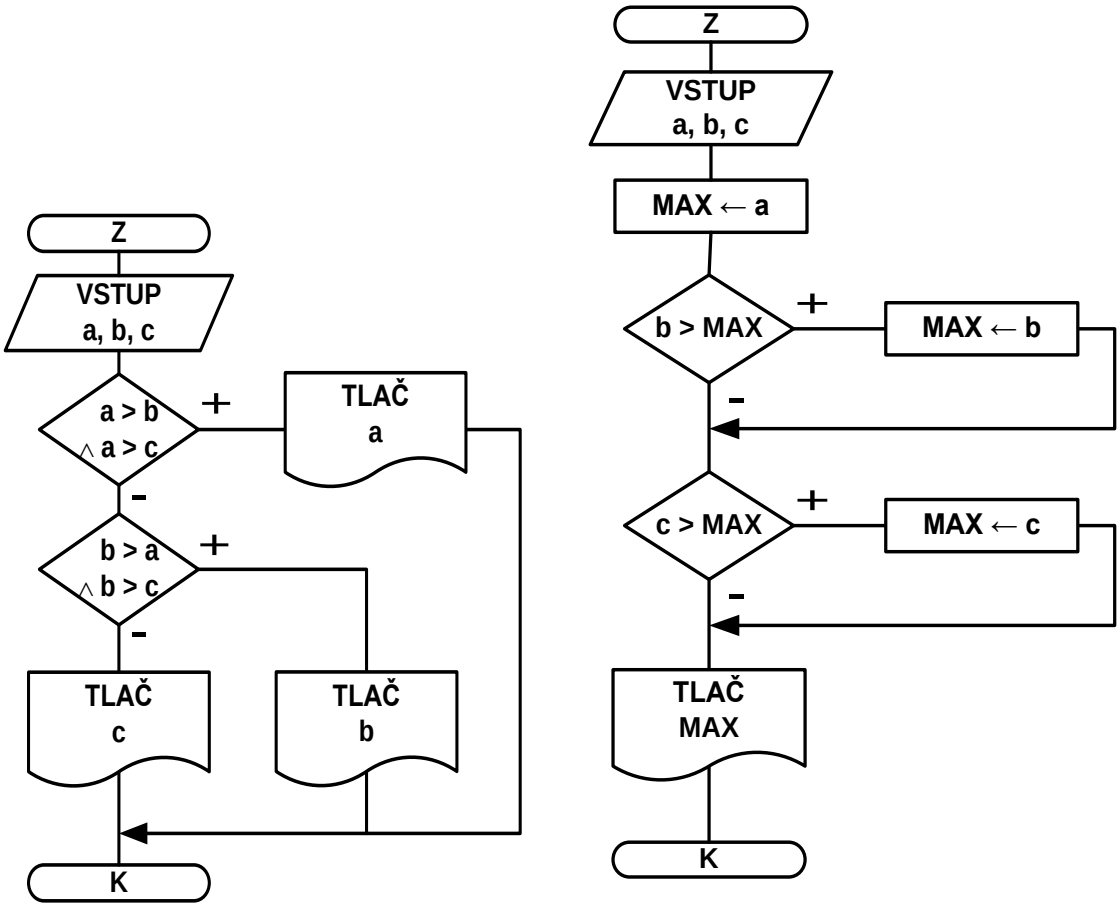


Obr. 3.3 b) VD – výpočet koreňov kvadratickej rovnice

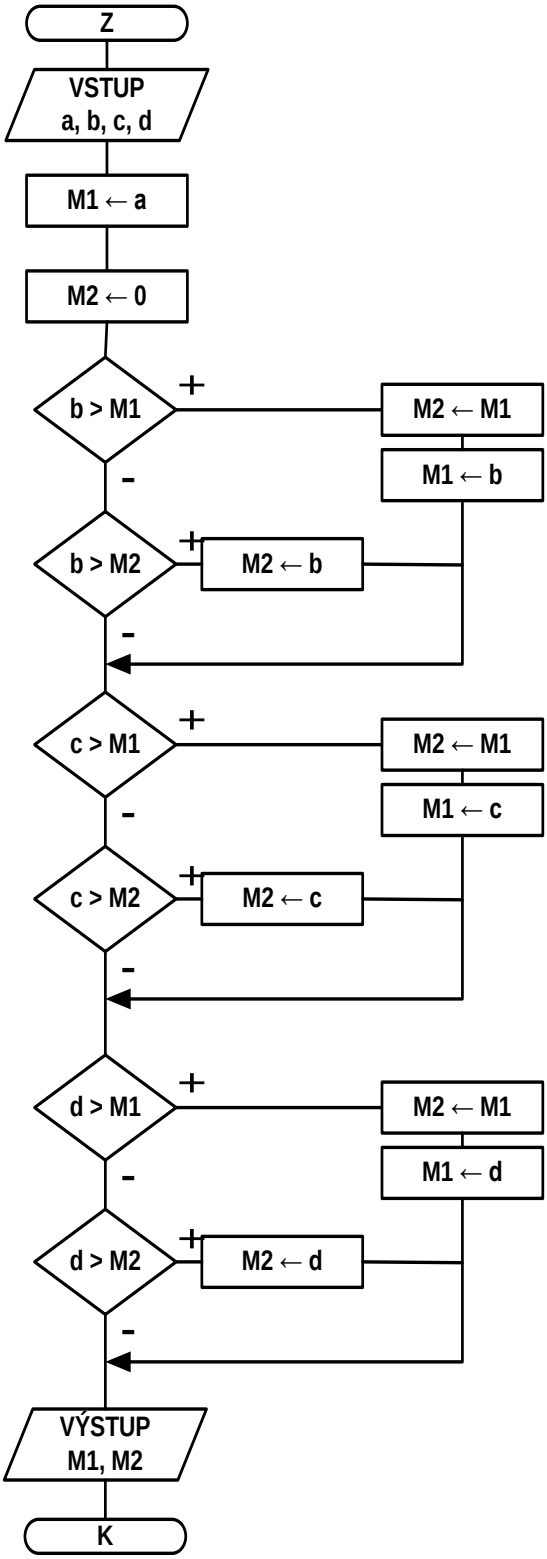
1.1.1 Vzorové príklady

Príklad 1 Z troch zadaných čísel označených a, b, c vyhľadajte a vytlačte najväčšie. Predpokladajme, že čísla sú rôzne, t.j. a ≠ b ≠ c.

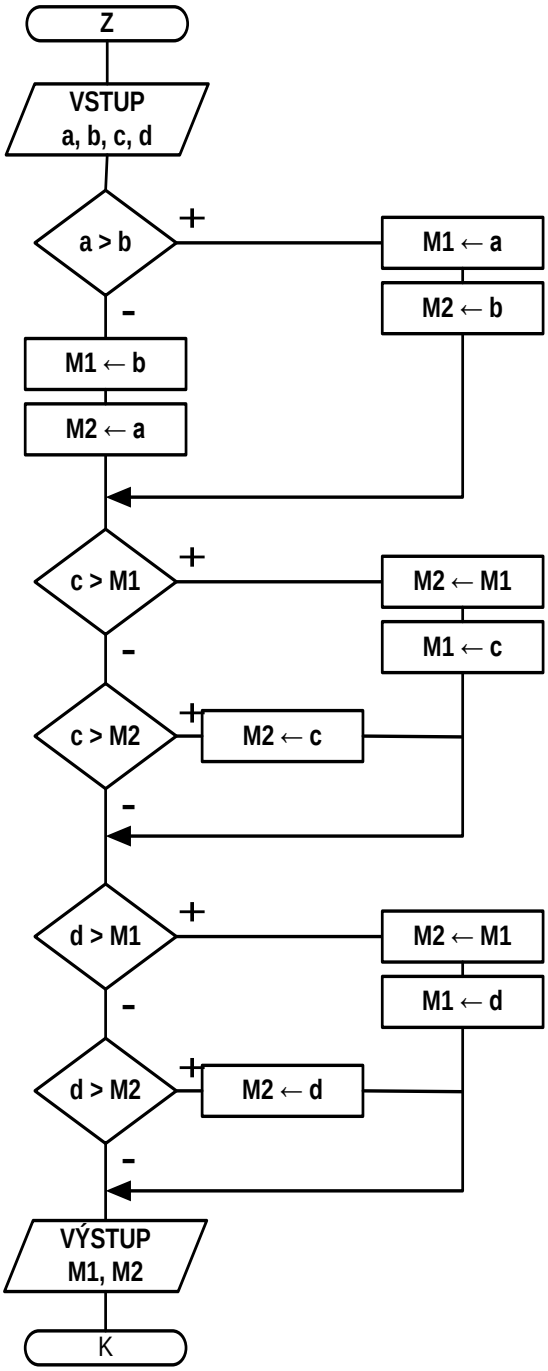




Príklad 2 Zo štyroch čísel označených a, b, c, d nájdite a vytlačte dve najväčšie. (Predpokladajme, že čísla sú rôzne, t.j. $a \neq b \neq c \neq d$).



V obore kladných čísiel

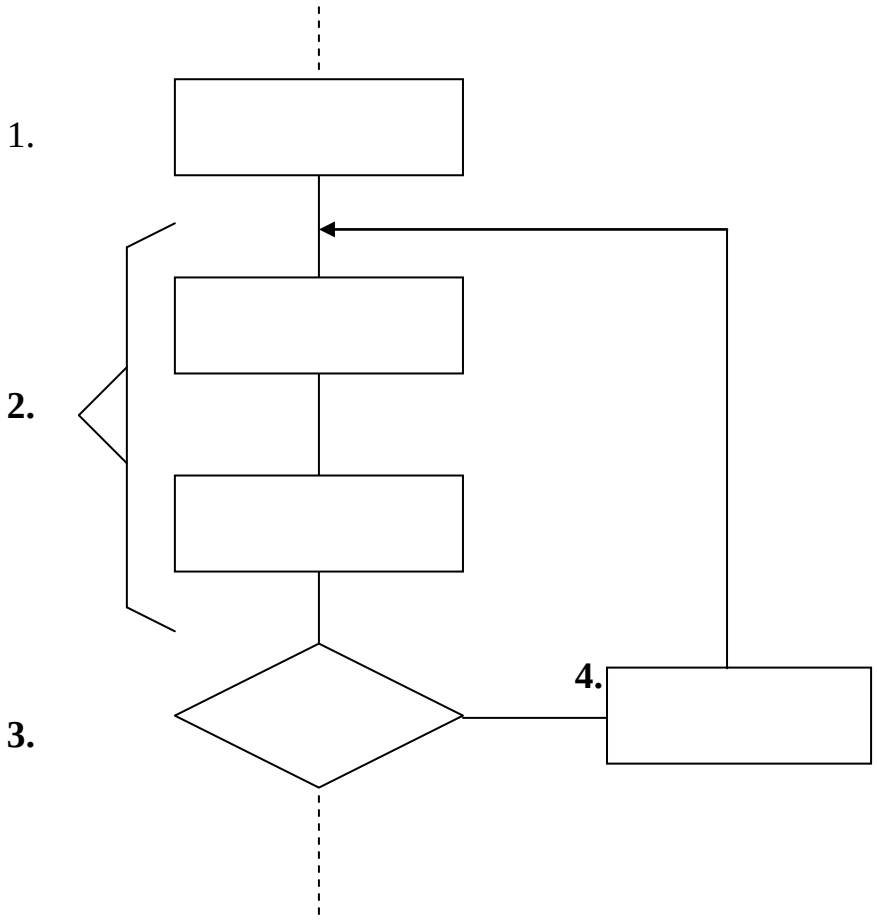


V obore reálnych čísiel

Vývojový diagram programu s vetvením s opakovaním

Pri predchádzajúcich typoch vývojových diagramov je približne toľko značiek, koľko sa má urobiť operácií. Ak by však išlo o veľké množstvo spracovávaných údajov, vznikali by ťažkosti pri riešení úloh takýmto spôsobom (dlhé programy, nedostatočná kapacita pamäti a pod.). Preto je potrebná iná forma zostavovania vývojových diagramov, ktorá by niektorú časť vývojového diagramu skrátila, t. j. napr. zopakovala toľkokrát, koľkokrát to riešenie vyžaduje. Tejto podmienke zodpovedajú vývojové diagramy s cyklom. Pri tomto riešení sa po vhodnej úprave opakuje určitá skupina operácií. V tomto prípade je počet uskutočňovaných operácií podstatne väčší ako počet značiek vývojového diagramu.

Cyklus vo vývojovom diagrame má štyri časti:
1. prípravná časť cyklu,
2. operačná (vnútorná) časť cyklu,
3. rozhodovacia časť cyklu (koncová podmienka),
4. modifikačná časť cyklu.



Obr. 3.1 Cyklus vo vývojovom diagrame

Parametrom cyklu môže byť jednoduchá alebo indexovaná premenná, ktorá môže nadobúdať hodnoty v určitých hraniciach, vymedzených rozsahom riešenej úlohy. V prípravnej časti cyklu sa priradí parametru počiatočná hodnota, ktorú parameter môže nadobudnúť. Operačná časť cyklu obsahuje operácie, ktoré sa majú opakovať, pričom uskutočnenie príslušnej operácie závisí od hodnoty parametra cyklu. Táto sa mení po každom uskutočnení všetkých operácií vo vnútri cyklu. Modifikácia parametra sa uskutočňuje pričítaním konštantného čísla (tzv. jednotka kroku) k parametru cyklu. V rozhodovacej časti cyklu sa rozhoduje o tom, či sa má pokračovať v príslušnom cyklickom výpočtovom procese, alebo nie. Rozhodovanie sa uskutočňuje porovnávaním hodnoty parametra s jeho hornou (dolnou) hranicou.

Vývojový diagram s cyklom uvedieme na nasledujúcom príklade:

Na vstupnom médiu sú zadávané jednotlivé prvky n – rozmerného vektora a, pričom prvý údaj predstavuje jeho rozsah n (počet prvkov vektora). Máme zostaviť vývojový diagram pre výpočet súčtu jednotlivých prvkov podľa vzorca:

$$S = \sum_{i=1}^{n} a_i$$

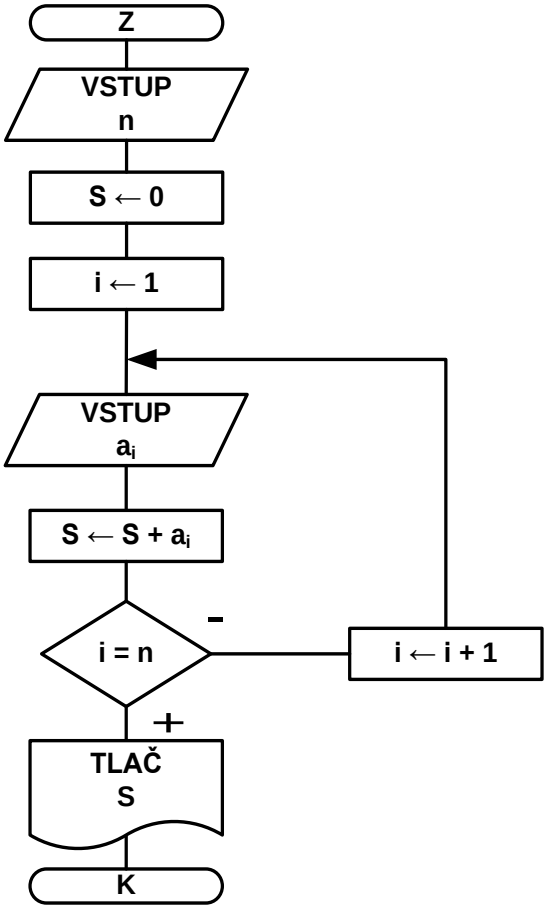
Celý postup riešenia možno rozvrhnúť do niekoľkých krokov. V tomto prípade vhodnejšie než zosnímanie všetkých údajov naraz, je postupné snímanie údajov so súčasným spracovaním (sčítavaním). Jednotlivé kroky budú nasledovné:

1. určíme a pripravíme si označenie výsledku,
2. zosnímame údaj o počte prvkov vektora,
3. určíme parametru cyklu dolnú hranicu,
4. zosnímame spracovávaný údaj,
5. údaj pripočítame k doterajšiemu súčtu,
6. zistíme, či bol zosnímaný posledný údaj (hodnotu parametra cyklu porovnáme s jeho hornou hranicou),
7. ak nie, parameter zvýšime o jednotku kroku a pokračujeme krokom 4,
8. ak áno, tlačíme výsledok.

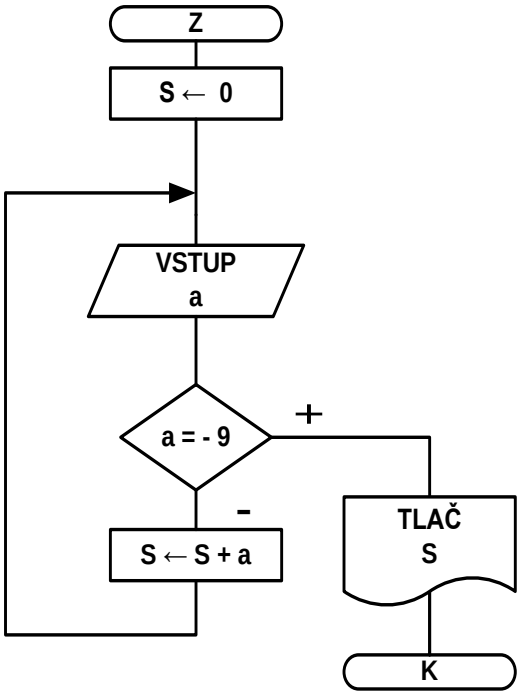
Vývojový diagram je uvedený na obr. 3.9.

V tomto prípade na začiatku poznáme počet spracovávaných údajov (počet prvkov vektora – n).

Údaje môžu byť zaznamenané i tak, že nepoznáme ich počet, ale poznáme ich zakončenie tzv. *koncový znak*. Je to znak, ktorý sa odlišuje od ostatných spracovávaných údajov. V tomto prípade nie je možné orientovať sa podľa indexu (parametra), pretože nie sú stanovené hranice cyklu. Údaje snímame taktiež postupne, až po koncový znak a tomu zodpovedá i otázka v rozhodovacej operácii, na základe ktorej sa určuje ďalší postup riešenia úlohy. Vývojový diagram má iný postup, ale riešenie – výsledok je samozrejme rovnaký. Ukážeme si to na príklade, kde máme úlohu ako v predošlom príklade (teda súčet prvkov), ale nevieme počet spracovávaných údajov. Údaje sú zakončené koncovým znakom „– 9“. Vývojový diagram je uvedený na obr. 3.10.



Obr. 3.2

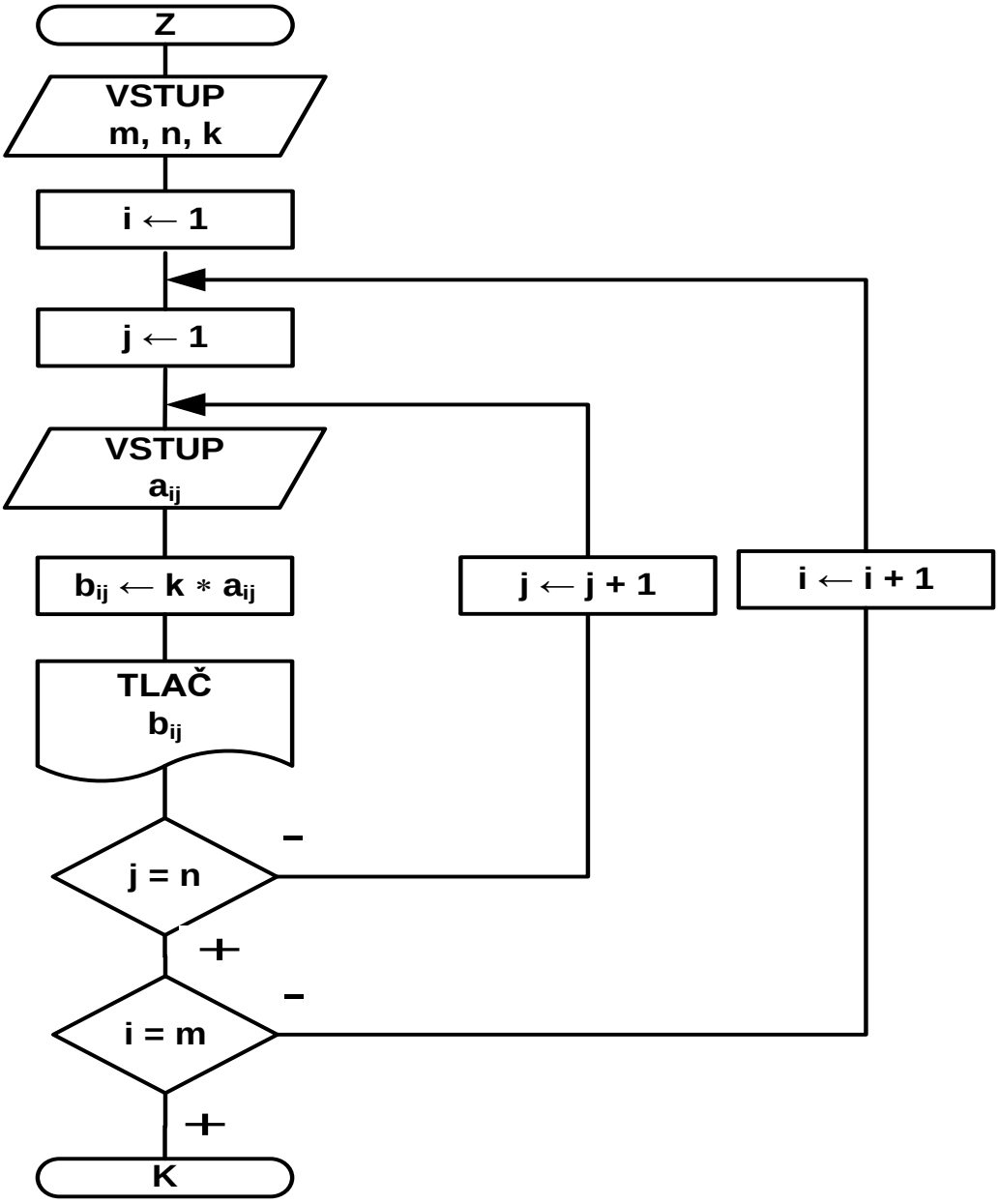


Obr. 3.3

Podľa riešenej úlohy sa môže vo vývojových diagramoch vyskytnúť i niekoľko cyklov za sebou, poprípade tzv. „vnorené“ cykly (cyklus v cykle). Príkladom je napr. úloha pre násobenie matice konštantou.

Na vstupnom zariadení sú zadávané prvky matice **A**(a_{ij}) i=1,2,...,m, j=1,2,...,n po riadkoch, pričom ako prvé tri údaje sú zadané údaje vyjadrujúce počet riadkov, počet stĺpcov a konštanta, ktorou máme maticu A vynásobiť podľa vzorca **B = k *A**

Vývojový diagram je uvedený na obr. 3.11.



Obr. 3.4 VD – násobenie matice konštantou

Matica **A** má *m* riadkov a *n* stĺpcov.

Pri spracovávaní prvkov matice vystupujú dva parametre cyklu:

i – pre označenie riadku a

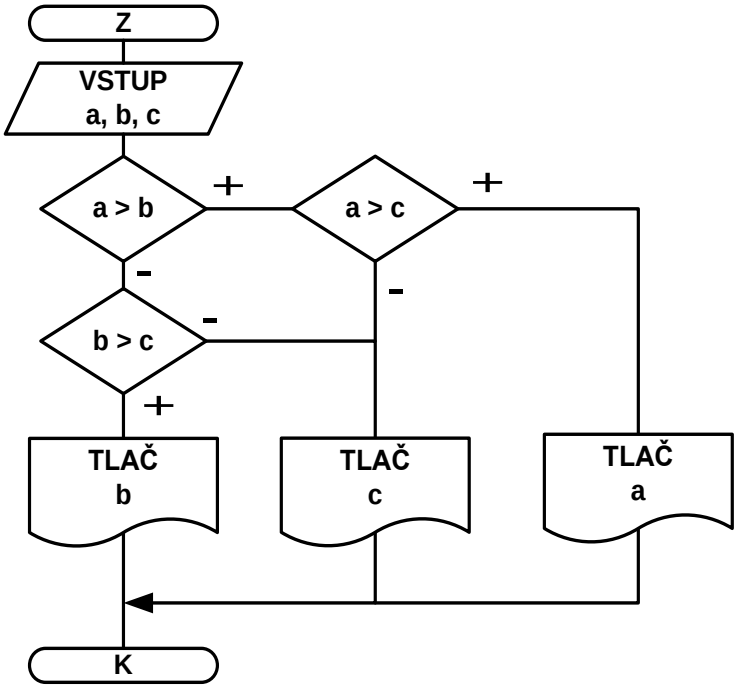
j – pre označenie stĺpca.

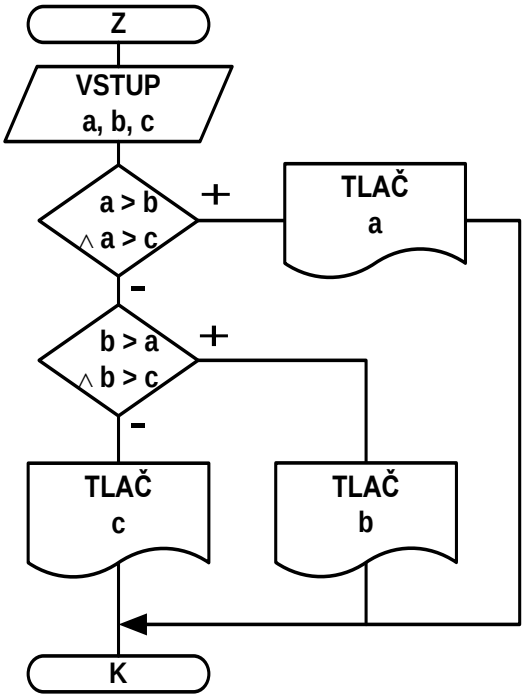
Každý prvok matice a_{ij} vyjadruje jeho umiestnenie v matici – v i-tom riadku a j-tom stĺpci.

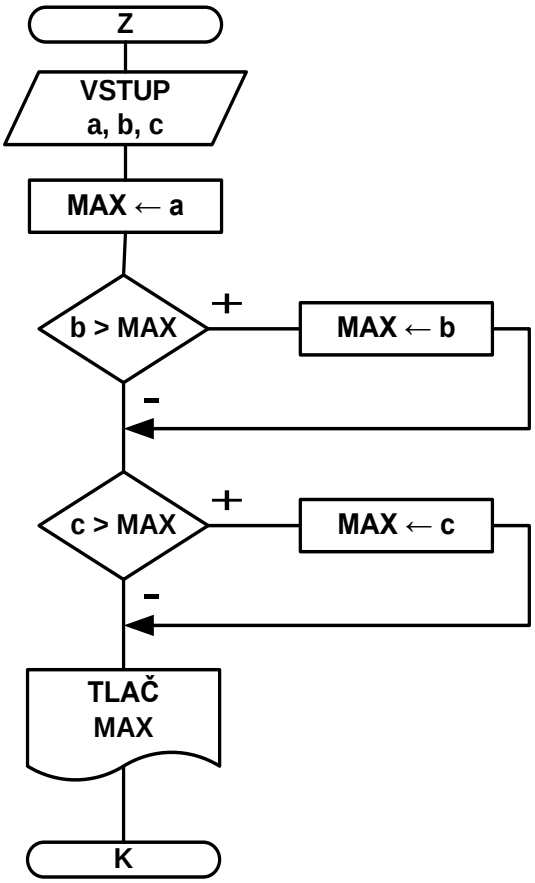
Ak prvky matice zadávame po riadkoch, rýchlejšie sa mení index *j* – pre označenie stĺpcov (preto musí byť ako vnútorný cyklus). Cyklus pre index *i* – pre označenie riadkov je vonkajší.

1.1.1 Vzorové príklady

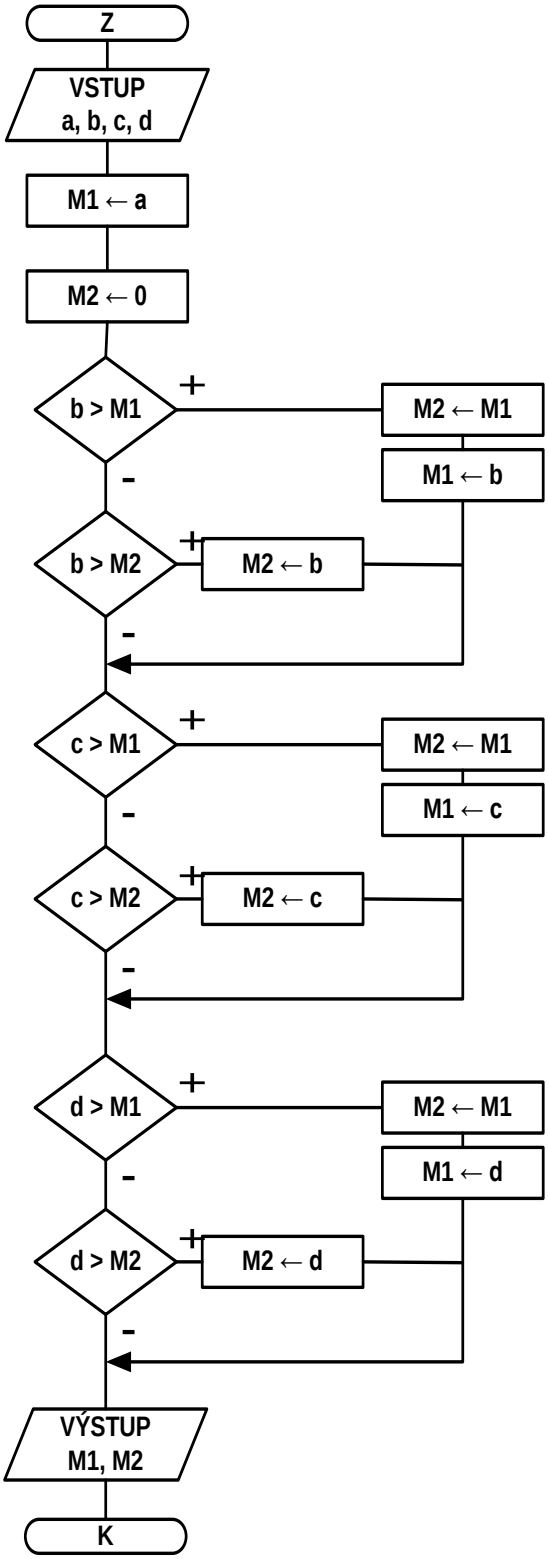
Príklad 1 Z troch zadaných čísel označených a, b, c vyhľadajte a vytlačte najväčšie. Predpokladajme, že čísla sú rôzne, t.j. $a \neq b \neq c$.



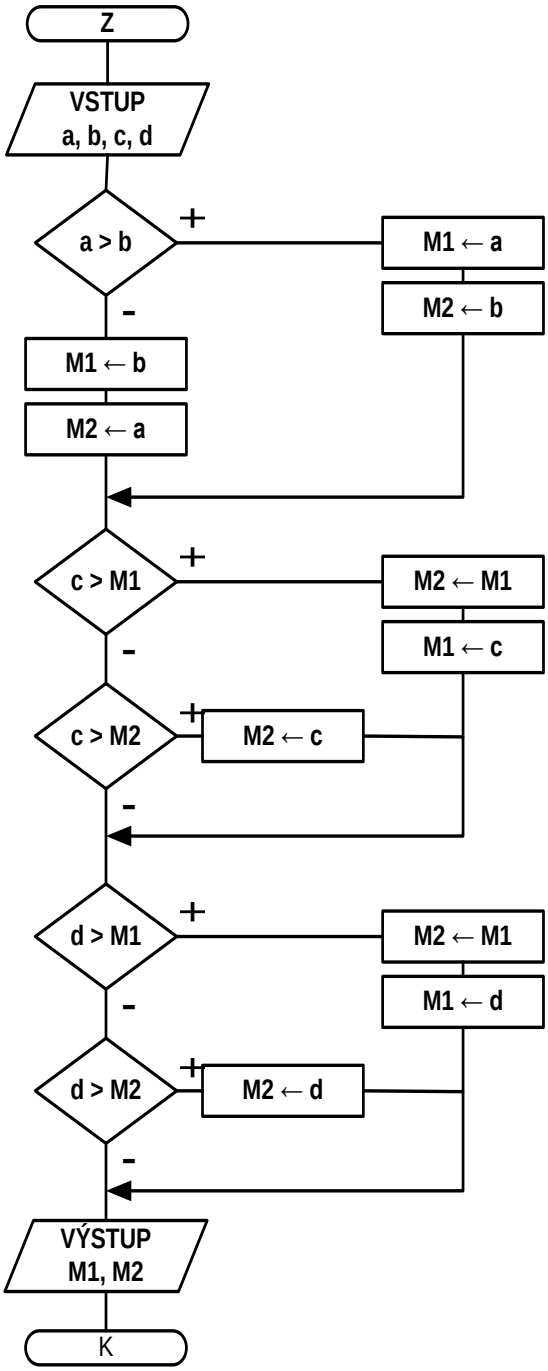




Príklad 2 Zo štyroch čísel označených a, b, c, d nájdite a vytlačte dve najväčšie. (Predpokladajme, že čísla sú rôzne, t.j. a ≠ b ≠ c ≠ d).

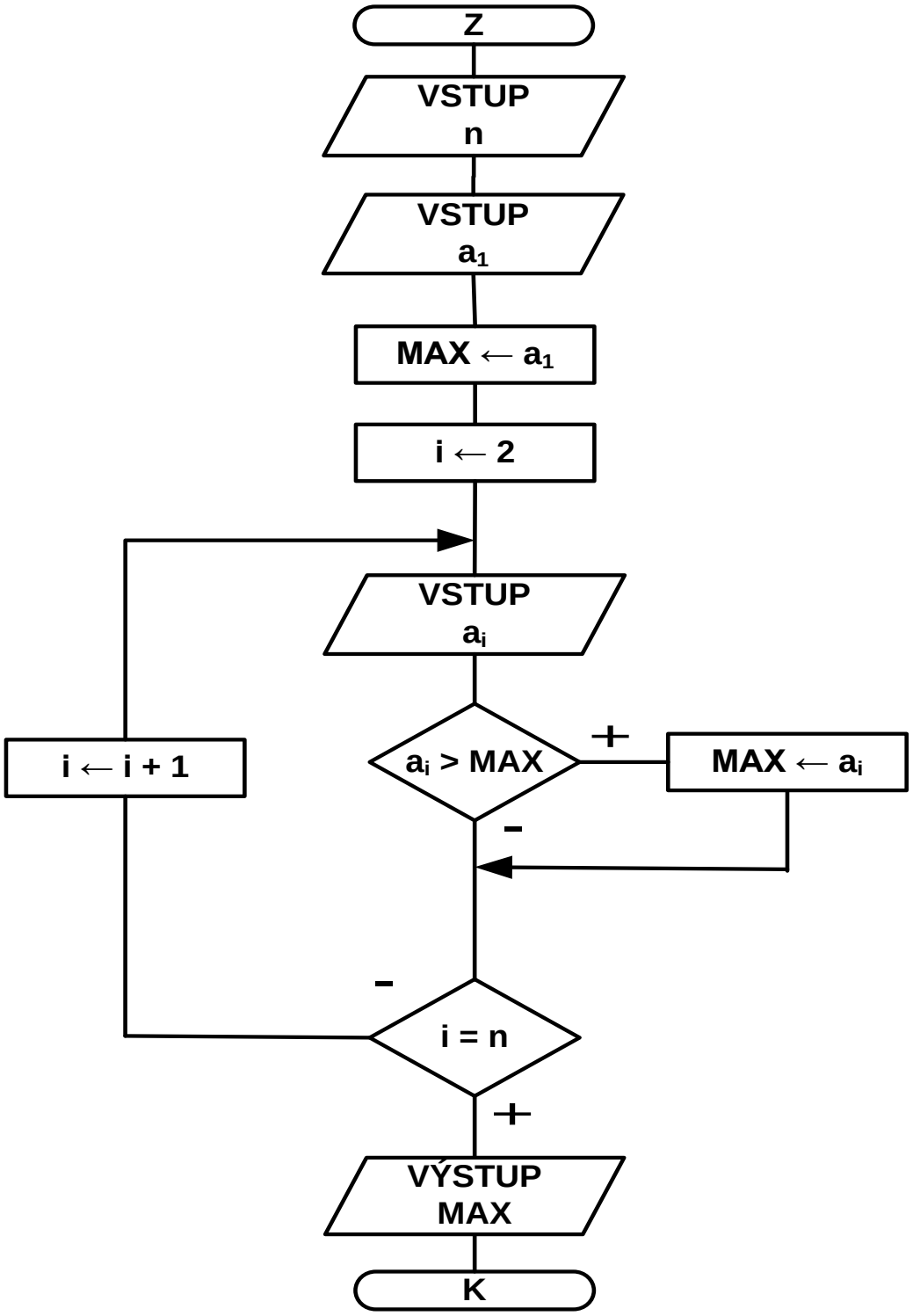


V obore kladných čísiel

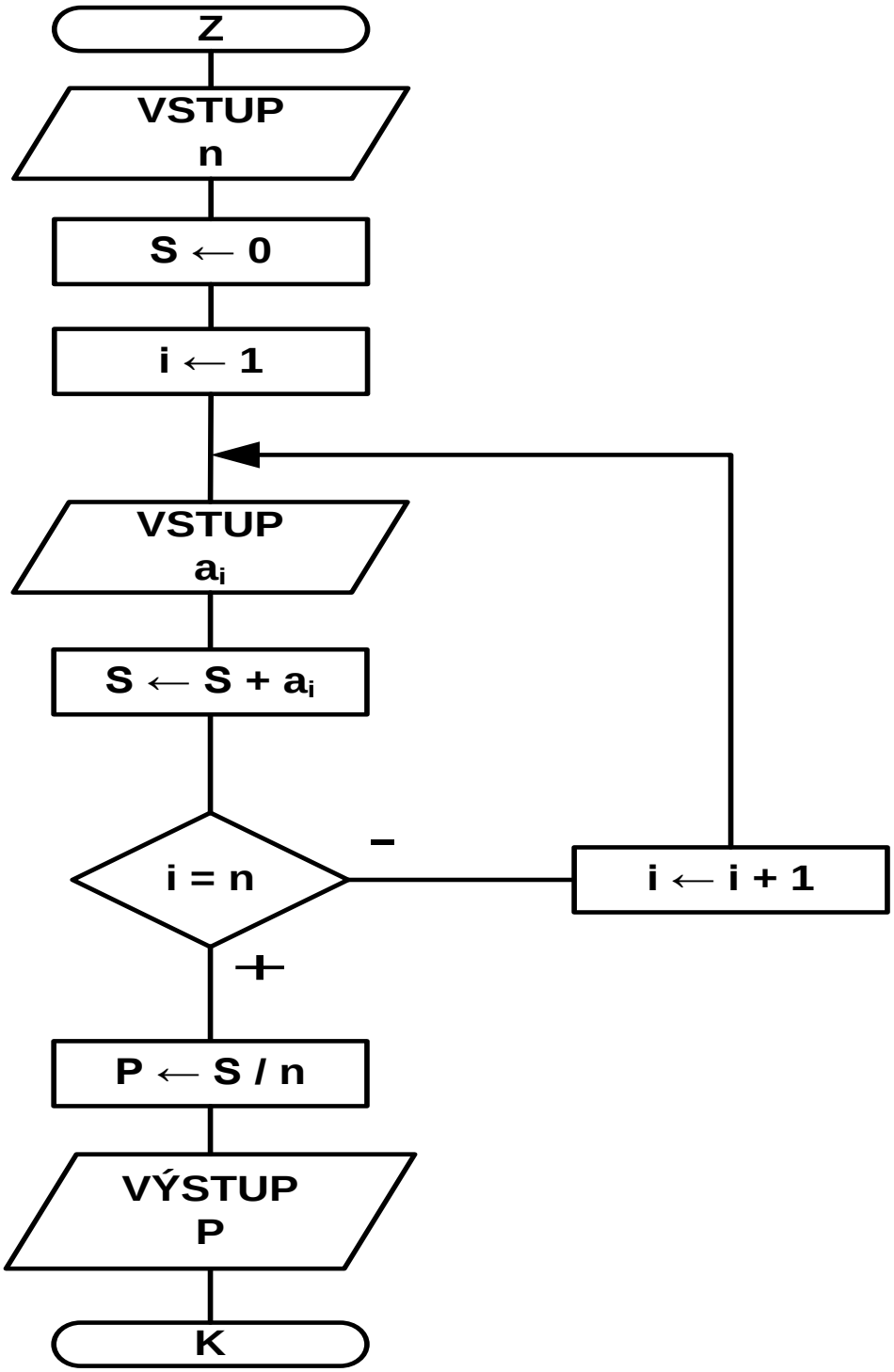


V obore reálnych čísiel

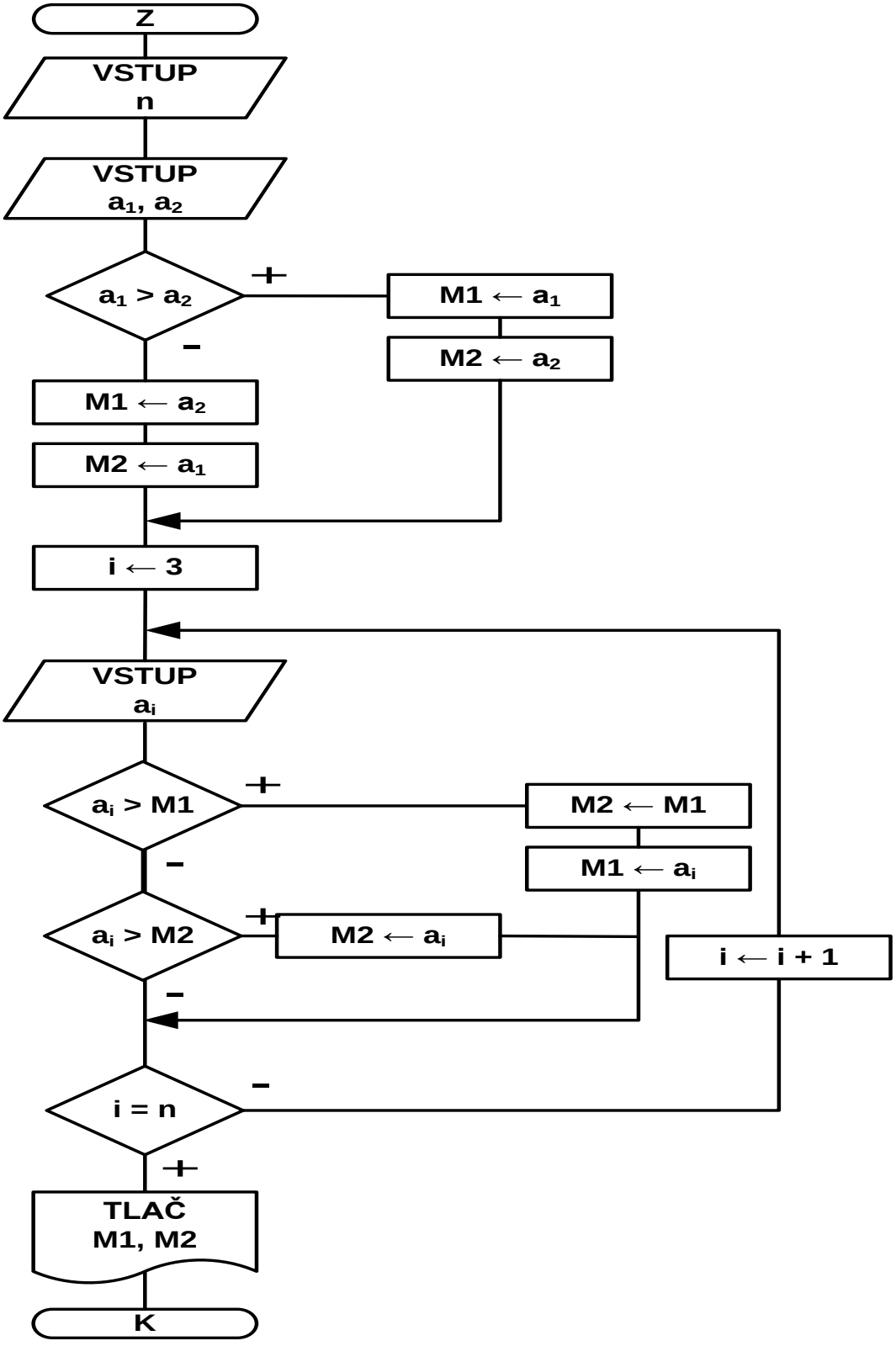
Príklad 3 Sú postupne zadávané hodnoty prvkov vektora $\vec{a}(a_1, a_2, ..., a_n)$ v poradí:
n – počet prvkov vektora,
$a_1, a_2, ..., a_n$ – prvky vektora,
Vyhľadajte a vytlačte najväčšiu hodnotu použitím pomocnej bunky **MAX**.



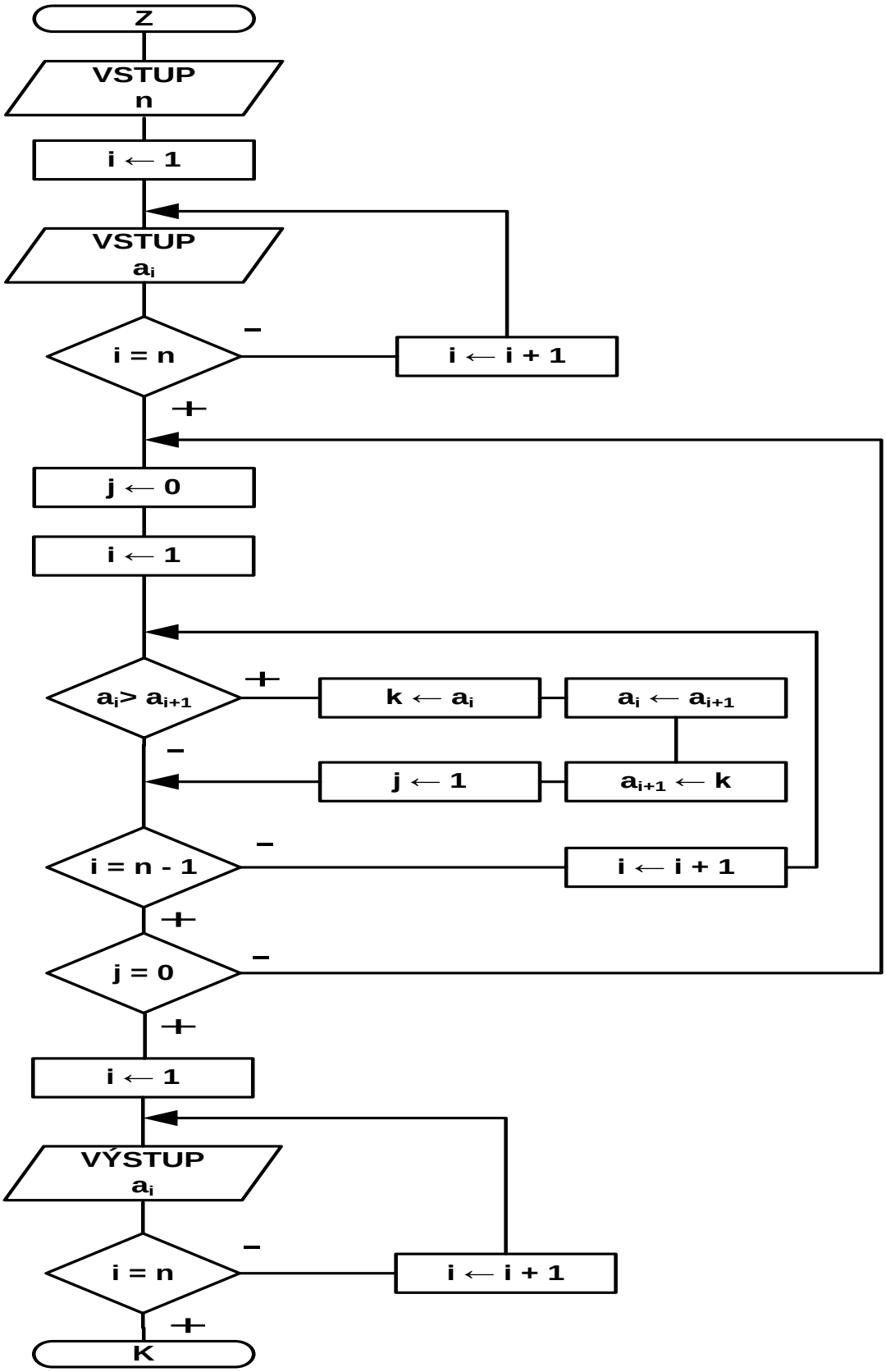
Príklad 4 Sú postupne zadávané hodnoty prvkov vektora $\vec{a}(a_1, a_2, ..., a_n)$ v poradí:
n – počet prvkov vektora,
a_1, a_2,..., a_n – prvky vektora.
Vypočítajte a vytlačte priemer z prvkov vektora označený **P**.



Príklad 5 Sú postupne zadávané hodnoty prvkov vektora $\vec{a}(a_1, a_2, ..., a_n)$ v poradí:
n – počet prvkov vektora,
a_1, a_2,..., a_n – prvky vektora.
Zostrojte vývojový diagram na vyhľadajte a vytlačenie dvoch najväčších hodnôt označených **M1**, **M2**.

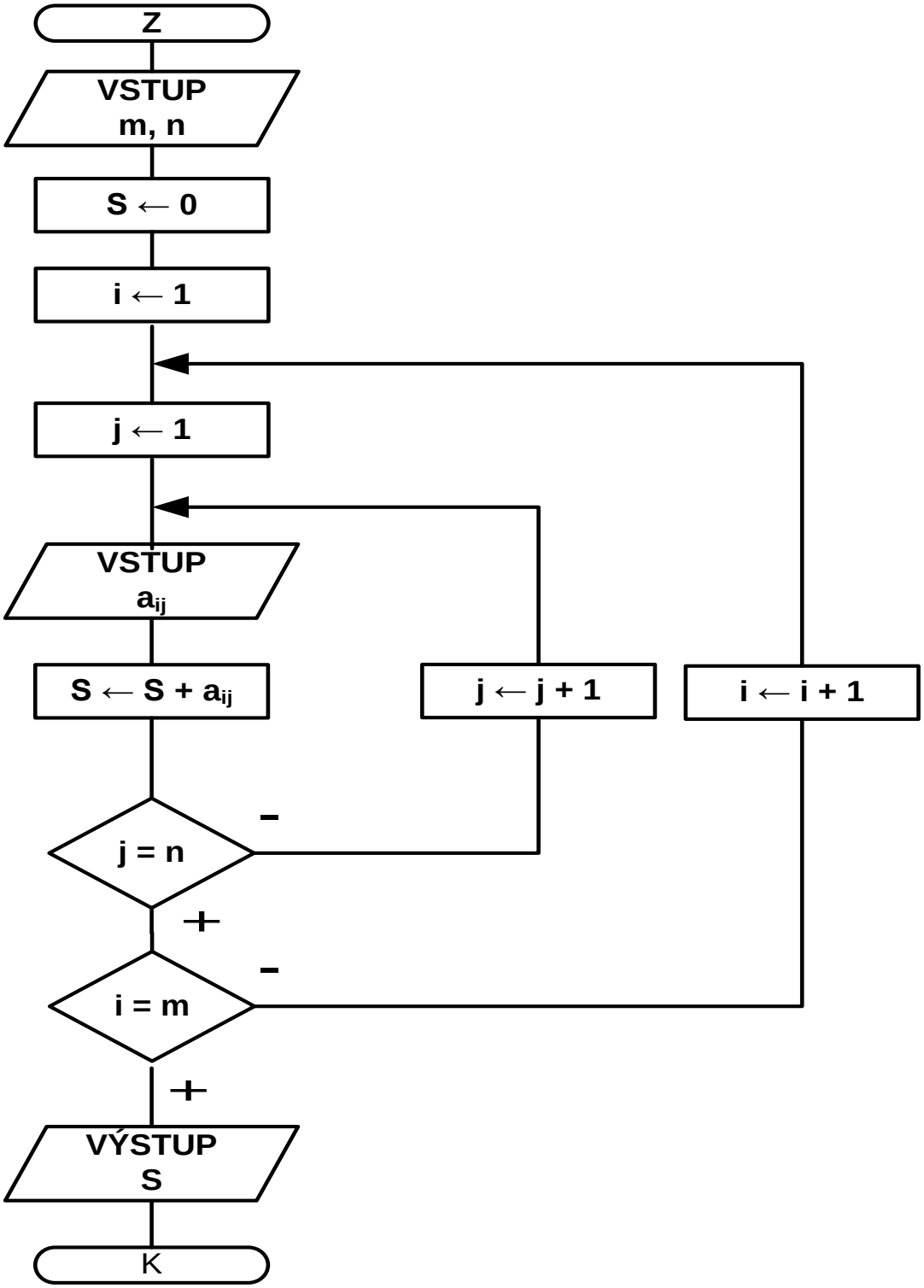


Príklad 6 Prvky vektora $\vec{a}(a_1, a_2, ..., a_n)$ sú zadávané na vstupe v poradí:
n – počet prvkov vektora, a_1, a_2,..., a_n – prvky vektora.
Zostrojte VD na usporiadanie prvkov vektora **vzostupne** (od najmenšieho po najväčší).



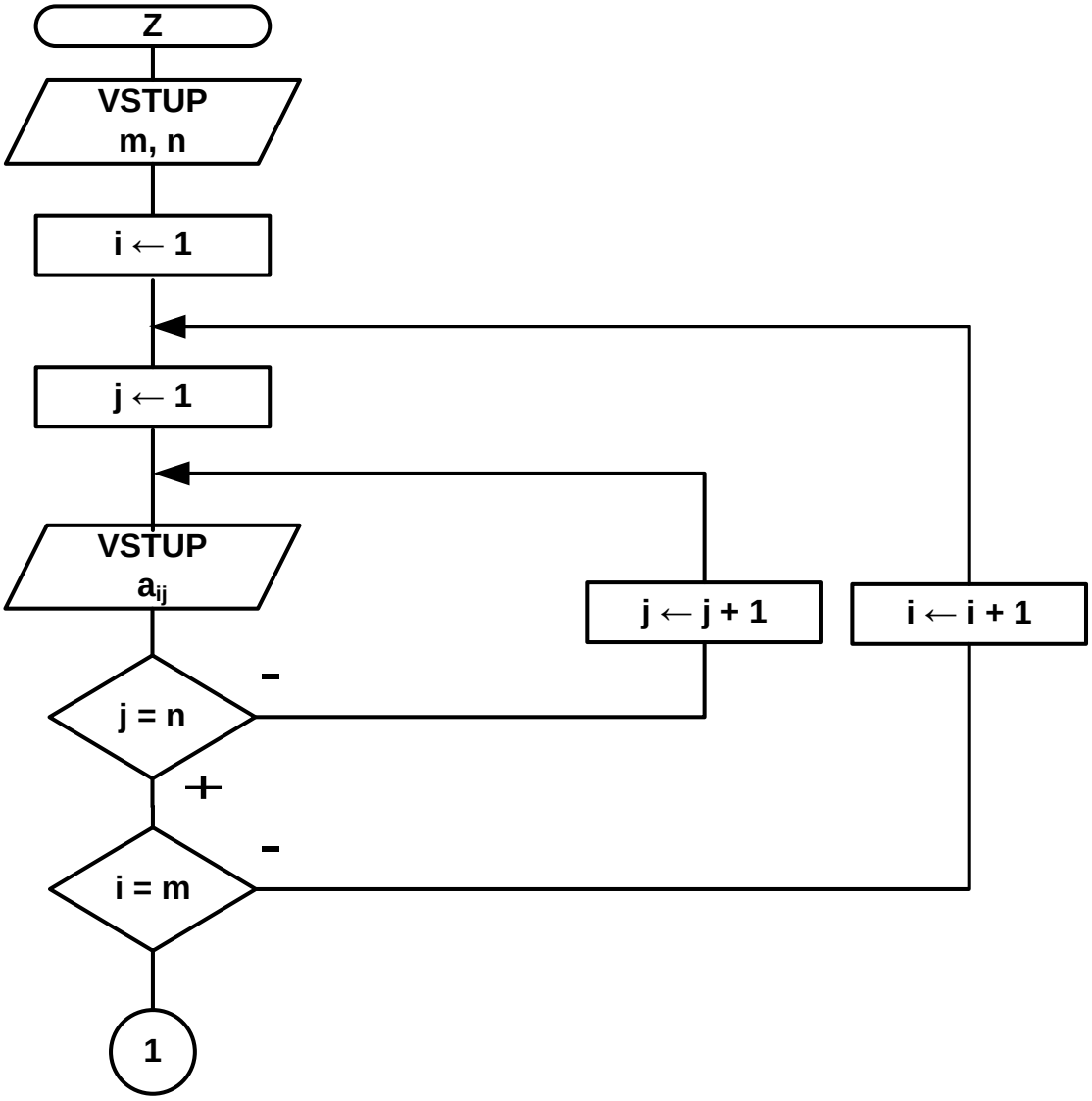
Príklad 7 Prvky matice $A(a_{ij})$ $i=1,2,...,m$, $j=1,2,...,n$ sa zadávajú po riadkoch. Zostrojte vývojový diagram na výpočet súčtu všetkých prvkov matice označený **S**.
Vstupné údaje:

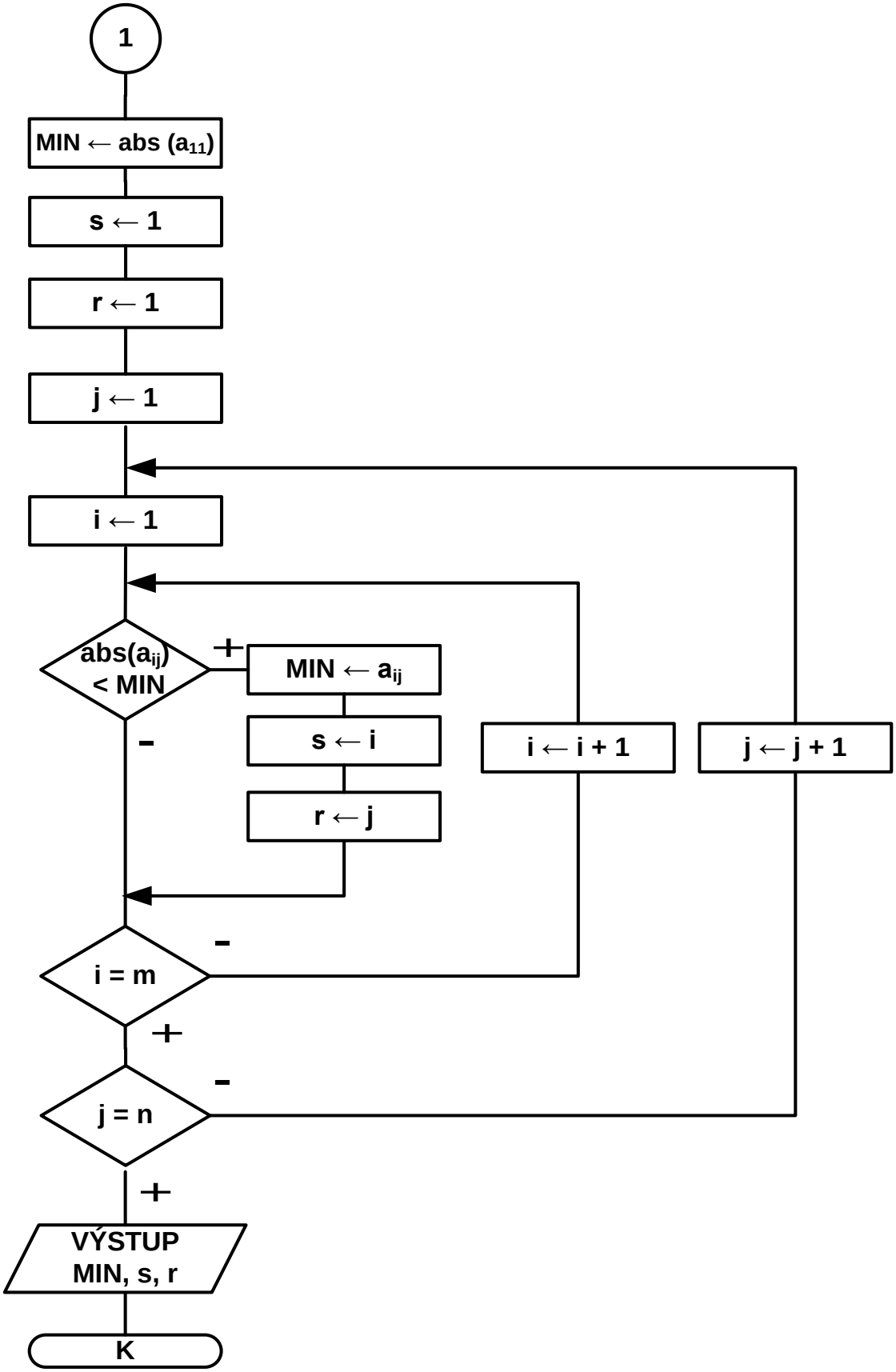
m – počet riadkov matice,
n – počet stĺpcov matice,
a_{ij} – prvky matice po riadkoch



Príklad 8 Zostavte algoritmus pre vyhľadanie a výstup **najmenšieho prvku v absolútnej hodnote** a jeho **indexov** z matice $A(a_{ij})$ $i=1,2,...,m$, $j=1,2,...,n$.
Vstupné údaje:

m – počet riadkov, *n* – počet stĺpcov, a_{ij} – prvky matice po riadkoch



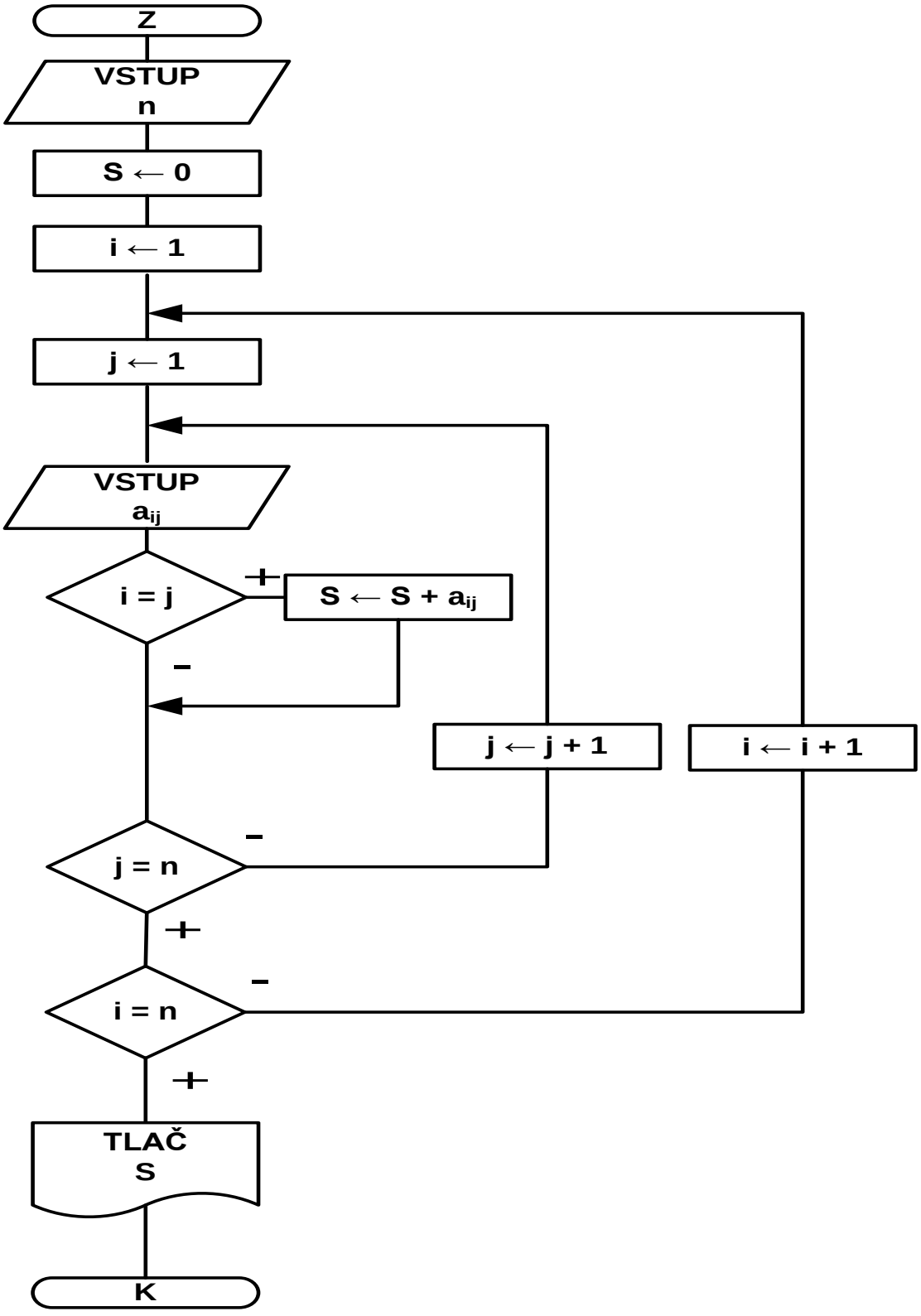

1
MIN ← abs (a11)
s ← 1
r ← 1
j ← 1
i ← 1
abs(aij) < MIN
+
MIN ← aij
s ← i
r ← j
-
i ← i + 1
j ← j + 1
i = m
-
+
j = n
-
+
VÝSTUP MIN, s, r
K

Príklad 9 Zostavte algoritmus pre výpočet súčtu prvkov na hlavnej diagonále štvorcovej matice $A(a_{ij})$ $i=1,2,...,n$, $j=1,2,...,n$.

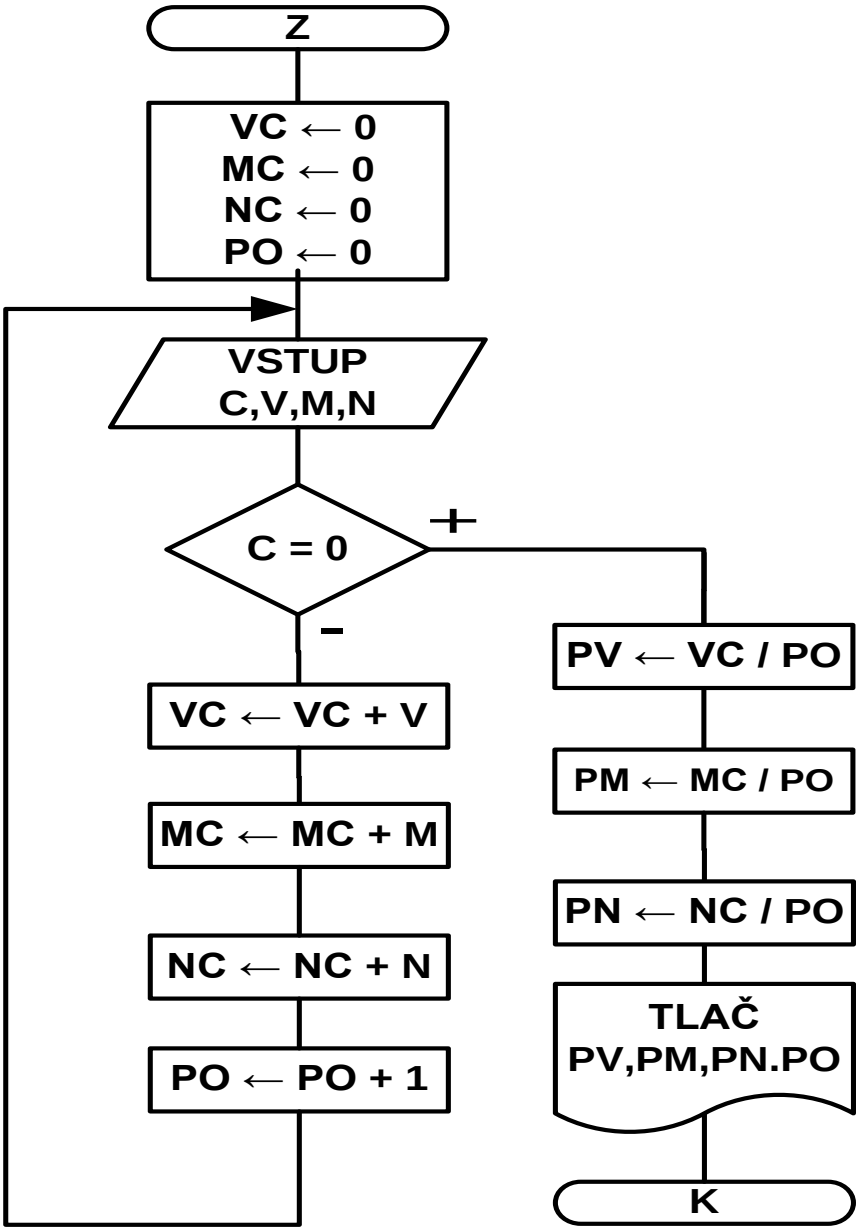
Vstupné údaje:

n – počet riadkov, počet stĺpcov,

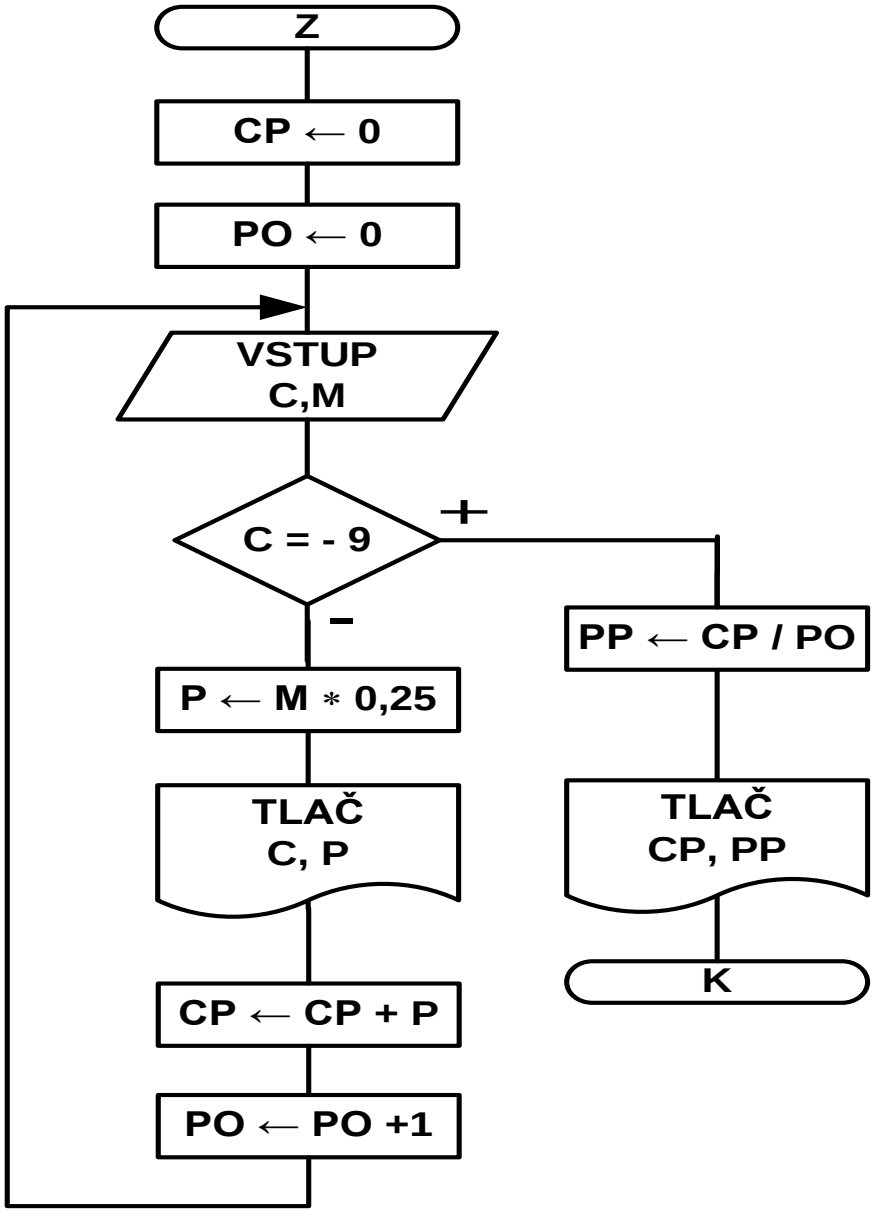
a_{ij} – prvky matice po riadkoch.



Príklad 10 Na vstupnom médiu sa zadávajú údaje o zamestnancoch podniku:
C – číslo zamestnanca,
V – vek,
M – mzda,
N - % plnenia normy.
Súbor údajov je ukončený koncovým znakom *C=0*.
Zostavte VD na výpočet a tlač:
PV – priemerného veku zamestnancov,
PM – priemernej mzdy,
PN – priemerného plnenia normy,
PO – počtu zamestnancov.



Príklad 11 Na vstupnom médiu sa zadávajú dvojice údajov o zamestnancoch podniku:
C – číslo zamestnanca,
M – hrubá mzda,
Súbor údajov je ukončený koncovým znakom *C=–9*.
Zostavte VD na výpočet a tlač:
P – výšky prémie za každého zamestnanca, ktorá je 25 % z hrubej mzdy,
CP – celkovú čiastku prémií za celý podnik (za všetkých zamestnancov spolu),
PP – priemernú výšku prémií pripadajúcu na jedného zamestnanca.



Príklad 12 O každom materiáli na sklade sa zadávajú údaje:
C – číslo materiálu,
JC – jednotková cena,
M – skladové množstvo,
N – normatív množstva.
Súbor údajov je ukončený koncovým znakom *C=0*.
Zostavte VD na výpočet hodnoty materiálu na sklade **H** = ∑ **(JC*M)** a zistite počet **PN** – koľko druhov materiálov je podnormatívnych (M<N) a vytlačte.