Anotace

- Dynamické programování,
- Grafové algoritmy.

Problémy řešitelné vyplněním tabulky

- Faktoriál,
- přednášející jde do M1,
- počet platných uzávorkování pomocí n párů závorek,
- počet rozkladů přirozeného čísla na součet nerostoucích kladných celých čísel,
- Pascalův trojúhelník,
- nejdelší rostoucí podposloupnost,
- **pořadí násobení matic.**

Násobení matic

- Násobení matic není komutativní, ale je asociativní.
- Máme-li vynásobit několik matic, nemůžeme jejich pořadí měnit,
- můžeme součin všelijak závorkovat.
- Různá uzávorkování jsou různě výhodná.
- Které z nich je to nejvýhodnější?
- Předpokládáme, že máme abstraktní funkce zjistící ze zadaných rozměrů složitost součinu matic správných tvarů.

Násobení matic

- Některé násobení provedeme jako poslední.

Násobení matic

- Některé násobení provedeme jako poslední.
- To úlohu rozdělí na (nejvýše) dvě další instance.

Násobení matic

- Některé násobení provedeme jako poslední.
- To úlohu rozdělí na (nejvýše) dvě další instance.
- V každé (menší) instanci opět nějaké násobení provedeme jako poslední.

Násobení matic

- Některé násobení provedeme jako poslední.
- To úlohu rozdělí na (nejvýše) dvě další instance.
- V každé (menší) instanci opět nějaké násobení provedeme jako poslední.
- Ideální podhoubí pro rekurzi!

Násobení matic

- function slozitost(od,az_do:integer):longint;
- for i:=od to az_do-1 do begin
- slozitost:=slozitost(od,i)+ slozitost(i+1,az_do)+samotne_nasobeni;

Opět budeme zbytečně násobit stále to samé.
Opět se toho zbavíme stejně.

Násobení matic

- function slozitost(od,az_do:integer):longint;
- if cache[od,az_do]=0 then
- for i:=od to az_do-1
- cache[od,az_do]:=slozitost(od,i)+ slozitost(i+1,az_do)+samotne_nasobeni;
- slozitost:=cache[od,az_do];

Metoda

- Všechny tyto problémy jsme řešili stejně:

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,
- ten počítal často to samé, proto jsme mu doplnili cache.

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,
- ten počítal často to samé, proto jsme mu doplnili cache.
- Pokud se podíváme pořádně, na správném místě cache najdeme výsledek.

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,
- ten počítal často to samé, proto jsme mu doplnili cache.
- Pokud se podíváme pořádně, na správném místě cache najdeme výsledek.
- Potřebujeme tam tedy tu rekurzi?

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,
- ten počítal často to samé, proto jsme mu doplnili cache.
- Pokud se podíváme pořádně, na správném místě cache najdeme výsledek.
- Potřebujeme tam tedy tu rekurzi?
- Ne a můžeme se jí zbavit za předpokladu, že zjistíme, jak vyplňovat cache, tedy tabulku.

Metoda

- Všechny tyto problémy jsme řešili stejně:
- Navrhli jsme rekurzívní algoritmus,
- ten počítal často to samé, proto jsme mu doplnili cache.
- Pokud se podíváme pořádně, na správném místě cache najdeme výsledek.
- Potřebujeme tam tedy tu rekurzi?
- Ne a můžeme se jí zbavit za předpokladu, že zjistíme, jak vyplňovat cache, tedy tabulku.
- Tomu říkáme dynamické programování.

Odstranění rekurze

- Přednášející jde do posluchárny:

```
a[1]:=1;
a[2]:=2;
for i:=3 to n do a[i]:=a[i-1]+a[i-2];
```

- Pascalův trojúhelník:

```
for i:=0 to n do
      for j:=0 to i do
            p(i,j):=p(i-1,j-1)+p(i-1,j);
```

  a okrajové případy zvlášť!

- Závorkování: Cvičení

Násobení matic – poučení

- Tomuto říkáme *dynamické programování.*
- Zpravidla implementujeme pouze fázi vyplňování tabulky.
- Na tom je nepříjemné určovat, odkud vyplňování zahájit.
- Není tudíž špatné navrhnout si aspoň na počátku rekurzívní řešení a až pak rekurzi "rozbít":
- `cache[od,az_do]:=min{cache[od,i]+cache[i,az_do]+ samotne_nasobeni}`
- což stačí udělat cyklem.

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.
- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.
- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.
- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,
- topologické uspořádání, faktorová množina (vyšetřování komponent).

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.
- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,
- topologické uspořádání, faktorová množina (vyšetřování komponent).
- Modifikace problémů: Maximální kostra, nejdelší cesta – čím se liší?

Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).

Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).

Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).
- **Bellman-Fordův algoritmus:** $n - 1$-krát zopakujeme: Z každého vrcholu zkus "natáhnout" cestu po všech hranách z něj vycházejících (tedy zlepši případnou nalezenou cestu).

Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).
- **Bellman-Fordův algoritmus:** $n - 1$-krát zopakujeme: Z každého vrcholu zkus "natáhnout" cestu po všech hranách z něj vycházejících (tedy zlepši případnou nalezenou cestu).
- Algoritmus funguje i při záporném ohodnocení hran, nesmí ale být přítomna záporná kružnice (kružnice záporné délky).

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.
- Tedy ji vybavíme cachí v podobě třírozměrného pole...

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.
- Tedy ji vybavíme cachí v podobě třírozměrného pole...
- ... a zjistíme, že rekurzi vůbec nepotřebujeme, že postačí čtyři cykly v sobě...

Floyd – Warshallův algoritmus pseudokód

```
for a:=1 to n-1 do
   for u in vrcholy do
      for v in vrcholy do
         for w in sousedi(v) do
            cache[a,u,v]:=min(cache[a,u,v],cache[a-1,u,w]+
                 length(w,v));
```

Minimální kostra

- Vstup: Graf s ohodnocenými hranami

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setřiď hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setřiď hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setřiď hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.
- Jarníkův (Primův): Pěstování stromu: K dosud postavenému stromu přidej hranu z něj vycházející, která má nejnižší váhu.

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setřiď hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.
- Jarníkův (Primův): Pěstování stromu: K dosud postavenému stromu přidej hranu z něj vycházející, která má nejnižší váhu.
- Všechny algoritmy počítají to samé. Důkazy korektnosti budou příště.

Modifikace

- Hledání nejtěžší kostry...

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.
- Tudíž i nalezení nejkratší cesty v (potenciálně záporně) ohodnoceném grafu je těžké (NP-těžký problém):

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.
- Tudíž i nalezení nejkratší cesty v (potenciálně záporně) ohodnoceném grafu je těžké (NP-těžký problém):
- Sedí za ním schovaná Hamiltonskost, tedy hledání cesty délky $n - 1$:

Modifikace II

- Vytvoř úplný graf, za hranu v původním grafu přidej hranu s váhou -1,
  za nehranu přidej hranu s váhou 1.

Modifikace II

- Vytvoř úplný graf, za hranu v původním grafu přidej hranu s váhou -1,
  za nehranu přidej hranu s váhou 1.
- Zkus všechny dvojice: Má pro nějaké nejkratší desta váhu $-n+1$?

Modifikace II

- Vytvoř úplný graf, za hranu v původním grafu přidej hranu s váhou -1,
  za nehranu přidej hranu s váhou 1.
- Zkus všechny dvojice: Má pro nějaké nejkratší desta váhu $-n+1$?
- Těžkost problému spočívá v tom, že s jeho pomocí jsme schopni vyřešit jiný problém (který máme za těžký).

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.
- Algoritmus je až nečekaně jednoduchý:

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.
- Algoritmus je až nečekaně jednoduchý:
- Dokud neodebereme poslední vrchol, opakujeme:

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.
- Algoritmus je až nečekaně jednoduchý:
- Dokud neodebereme poslední vrchol, opakujeme:
- Najdi vrchol vstupního stupně 1 a přidej na konec dosud utvořeného lineárního uspořádání.

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.
- Algoritmus je až nečekaně jednoduchý:
- Dokud neodebereme poslední vrchol, opakujeme:
- Najdi vrchol vstupního stupně 1 a přidej na konec dosud utvořeného lineárního uspořádání.
- Pokud takový vrchol neexistuje (a zbývá neprázdný graf), ohlas, že to nejde.

Analýza algoritmu

- Korektnost algoritmu: Podmínka, kterou algoritmus testuje, je očividně postačující. A je také nutná, protože do vrcholu, který zařadíme před všechny ostatní, nesmí vést žádná hrana (má-li být větší, než všechny zbývající).

Analýza algoritmu

- Korektnost algoritmu: Podmínka, kterou algoritmus testuje, je očividně postačující. A je také nutná, protože do vrcholu, který zařadíme před všechny ostatní, nesmí vést žádná hrana (má-li být větší, než všechny zbývající).
- Konečnost a složitost algoritmu je též jasná: V každém kroku koukneme na každou hranu nejvýše dvakrát (má dva konce). Složitost je tedy $O(mn)$.

Faktorová množina

Alias večírek s roky narození

- Máme neorientovaný graf určující ekvivalenci, od které neznáme všechny prvky jsoucí v dotyčné relaci a chceme vyšetřit třídy ekvivalence.

Faktorová množina

Alias večírek s roky narození

- Máme neorientovaný graf určující ekvivalenci, od které neznáme všechny prvky jsoucí v dotyčné relaci a chceme vyšetřit třídy ekvivalence.
- Algoritmus: Každý vrchol má číslo. My mu přiřadíme ještě atribut "číslo komponenty", které bude pro začátek totožné.

Faktorová množina

Alias večírek s roky narození

- Máme neorientovaný graf určující ekvivalenci, od které neznáme všechny prvky jsoucí v dotyčné relaci a chceme vyšetřit třídy ekvivalence.
- Algoritmus: Každý vrchol má číslo. My mu přiřadíme ještě atribut "číslo komponenty", které bude pro začátek totožné.
- Následně jdeme přes všechny hrany a pro každou vyšetříme, zda mají její koncové vrcholy stejné číslo komponenty. Pokud ne, změníme **všechny** výskyty komponenty s vyšším číslem hodnotou komponenty s nižším číslem.

Faktorová množina

Alias večírek s roky narození

- Máme neorientovaný graf určující ekvivalenci, od které neznáme všechny prvky jsoucí v dotyčné relaci a chceme vyšetřit třídy ekvivalence.
- Algoritmus: Každý vrchol má číslo. My mu přiřadíme ještě atribut "číslo komponenty", které bude pro začátek totožné.
- Následně jdeme přes všechny hrany a pro každou vyšetříme, zda mají její koncové vrcholy stejné číslo komponenty. Pokud ne, změníme **všechny** výskyty komponenty s vyšším číslem hodnotou komponenty s nižším číslem.
- Nakonec si spočítáme třídy ekvivalence a postupně vypíšeme (znamenáme si u každého čísla, zda už bylo vypsáno a když najdeme prvek ještě nevypsaný, zaznamenáme si číslo komponenty a vypíšeme všechny prvky z této komponenty).

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`
- `funkcni_promenna=porovnej;`

Předání funkce parametrem

- Různé datové typy porovnáváme různě (string, integer).
- Chceme-li univerzální funkci, která bude třídit cokoliv se jí dostane do ruky, potřebujeme předat porovnávající funkci.
- Syntakticky postupujeme tak, že vytvoříme datový typ tuto funkci nesoucí.
- Do proměnné příslušného typu můžeme odpovídající funkci přiřadit a pak můžeme tuto funkci zavolat.
- Syntax: `type jmeno_typu=function (argumenty:typy):navratovy_typ;`
- `var funkcni_promenna:jmeno_typu;`
- `function porovnej(argumenty:typy);...`
- `funkcni_promenna=porovnej;`
- `funkcni_promenna(parametry);`

Příklad

```
type porovfce=function (a,b:integer):boolean;
var p:porovfce;
function cmp(a,b:integer):boolean;
begin
      cmp:=(a<b);
end;
procedure por(c:porovfce;a,b:integer);
begin
      if(c(a,b)) then writeln('Prvni vetsi!');
end;
begin
      porovnej:=cmp;
      if(p(10,20)) then writeln('Prvni vetsi');
      por(cmp,10,20);
end
```

Konec

...děkuji za pozornost...

Otázky?