Anotace

- Dynamické programování,
- předání funkce v parametru.

Nejdelší rostoucí podposloupnost

- Utvoř posloupnost dvojic (třeba pomocí pole recordů),
- první prvek obsahuje příslušnou hodnotu, druhý ukazuje délku nejdelší rostoucí podposloupnosti končící dotyčným prvkem.
- Vyplňuj zleva doprava, pro každý prvek najdi mezi prvky jemu předcházejícími takový menší prvek, ve kterém končí nejdelší dostupná podposloupnost.
- Podposloupnost najdi od konce:
- Najdi prvek nabízející největší možnou délku,
- poznamenej si HODNOTU a DÉLKU.
- postupuj od konce a pokud najedeš na prvek nabízející délku DÉLKA s hodnotou nejvýše tolik, kolik HODNOTA, prvek si poznamenej, sniž DÉLKU o jedna a HODNOTU na hodnotu nalezeného prvku.
- Nalezenou posloupnost otoč.

Nejdelší rostoucí podposloupnost – výpočet (vyplnění tabulky)

```
for i:=1 to n do begin
     maximum:=0; maxindex:=0;
     for j:=i-1 downto 1 do begin
          if pole[j].hodnota<pole[i].hodnota
                         and pole[j].delky>maximum then
          begin
               maximum:=pole[j].delky;
               maxindex:=j;
          end;
     pole[i].delky:=maximum+1;
end;
```

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.
- 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];

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.

Problém batohu

Definition

Problémem batohu nazveme problém, kdy máme zadány váhy jednotlivých předmětů $(w_1, \ldots w_n)$ a jejich ceny $(c_1, \ldots c_n)$ a nosnost batohu m a ptáme se, jak naložit batoh co nejcennějším obsahem.

- Předpokládejme variantu, že všechna čísla jsou přirozená!
- Rekurzivní řešení:
- Načti předměty a volej funkci přidej(k,l).
- První parametr říká, kolik předmětů už v batohu je, druhý určuje index posledního z nich.
- Nevýhoda: Zkoušíme všechny možnosti.
- Zlepšení? Dynamickým programem, ale neotřelým způsobem.

- Vytvoř pomocné batohy s nosnostmi $1, 2, \ldots m$ a proměnné popisující optimum v nich zinicializuj nulami.
- Pro každý předmět i přepočítej všechny batohy a zjisti, zda dopadnou lépe, pokud předmět přidáme nebo ne:
- Pro batoh k zjisti, pokud `cena_batohu(`k`)<cena_batohu(`$k - w_i$`)` $+ c_i$.
- Pokud ano, `cena_batohu(`k`):=cena_batohu(`$k - w_i$`)` $+ c_i$.
- Invariant: Řešíme-li prvek i, evidujeme (před touto fází) optimální naplnění batohů pomocí prvků $1, 2, \ldots k - 1$.
- Složitost: mn.
- Pozor, pokud nejsou čísla přirozená, stává se problém NP-těžkým. Není ale silně NP-těžký, což znamená, že těžkost se vynucuje "velkými čísly lišícími se o málo".

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
      p:=cmp;
      if(p(10,20)) then writeln('Prvni vetsi');
      por(cmp,10,20);
end
```