Anotace

- Řídké polynomy a matice,
- údržba paměti svépomocí,
- hashování.

Reprezentace spojovými seznamy

- Chceme reprezentovat řídké polynomy (tedy s málo nenulovými koeficienty). Jak to udělat?
- Pomocí spojových seznamů.
- Vhodný je obousměrný spojový seznam...
- ...a možná i cyklický – a v tom případě s hlavou!
- Sčítání polynomů: Prolezení spojových seznamů (podobně jako merge).
- Násobení polynomů: Potřebujeme lézt oběma směry.
- Hlava je k tomu, abychom poznali, že jsme na konci.

Reprezentace řídkých matic

- Opět máme málo nenulových prvků, matici tedy komprimujeme.
- Možností je mnoho, je třeba zejména přemýšlet (při použití). Například:
- Spojový seznam prvků (uspořádaný v obou dimenzích),
- spojový seznam spojových seznamů (řádky v jednom, sloupce ve druhém),
- "čtvrcení za živa", tedy rozdělíme matici na čtyři čtvrtiny, ty dělíme dále, až je ve "čtvrtině" málo prvků...

Funkce nízké úrovně

- `procedure mark(var p:pointer);` – oznámí vrchol haldy (kam až bylo allokováno)
  nepoužívat (radí kolega Kryl)
- `procedure release(var p:pointer);` – nastaví vrchol haldy (odallokuje všechno, co je nad vrcholem)
  nepoužívat (DTTO)
- Podle všeho ve FPC už nejsou, jsou nebezpečné, je to určitý pokus o garbage collection.
- `function MemAvail: longint;` – vrátí počet dostupných bytů na haldě
  (není ve Free Pascalu od verze 2.0)
- `function MaxAvail: longint;` – vrátí délku nejdelšího volného bloku (největší naalokovatelnou velikost)
  (DTTO)

Funkce nízké úrovně

- `GetMem` – naalokuje paměť, narozdíl od `new` nezkoumá kolik – používat jen v případě nouze (radí F. Kaempfl)
- `FreeMem` – odallokuje paměť naalokovanou pomocí `GetMem` – (DTTO)

Příklad použití GetMem/FreeMem

Vyrobíme pole předem neznámé délky

```
type ppole=^tpole;
     tpole=array[1..10000] of longint;
var pole:ppole;
begin
      GetMem(pole,500);{Urafni 500 bytu}
      pole^[10]:=1000;{To je OK}
      pole^[500]:=1024;{To je K. O., pole je male!}
      FreeMem(pole,500);{Melo by stacit i jen
FreeMem(pole);}
end.
```

Hashování

- Máme-li data, kterými lze indexovat, ale hodnoty by byly příliš velké (například řetězce), má smysl zkusit spočítat nějakou funkci (například posčítat ordinální hodnoty jednotlivých znaků mod 256). Tuto funkci označíme h.
- Uděláme tak tabulku mnohem menší velikosti, než je universum.
- Tomu říkáme hashování.
- Jenže více prvků může kandidovat na totéž místo tabulky, tomu říkáme kolize.

Hashování

řešení kolizí

- Různé metody řešení kolizí:
- Uděláme spojový seznam,
- srůstající hashování, tedy umísťujeme prvky do přihrádek, kam nepatří,
- z toho plyne problém s implementací `delete` (mazat prakticky nelze, protože bychom mohli roztrhnout řetízek).
- Na to kam umístit prvek v kolizi existuje mnoho metod. Buďto vybereme další volné místo, nebo obstaráme funkci, která určí další kandidátské místo.
- Pokud známe množství dat, můžeme se pokusit o perfektní hashování, a to do pole kvadraticky velkého (se středním počtem kolizí menším než 1), nebo dokonce do pole velikosti $4n$, ale pak předem musíme znát data.

Konec

Děkuji za pozornost...