Anotace

- Dijkstrův algoritmus,
- medián lineárně,
- odstranění rekurze,
- binární soubory,
- předání funkce v parametru.

Středník

3. května od 15:00 do 18:00.

Dijkstrův algoritmus

Hledá nejkratší cestu z daného vrcholu (do všech ostatních)

Vstup: Graf s `nezáporně` ohodnocenými hranami.

- Udržujeme "frontu" vrcholů setříděnou podle dosud nejkratší cesty do nich.
- Na začátku zinicializujeme vzdálenosti do všech vrcholů kromě startovního nekonečnem (tedy dost vysokou hodnotou) a vzdálenost do startu nulou.
- Startovní vrchol přidáme do "fronty" dosažitelných vrcholů.
- Vrchol, do kterého se dostaneme nejkratší cestou z fronty odstraníme a pokusíme se cestu jdoucí z něj rozšířit do jeho sousedů.
- Toto opakuj, dokud je "fronta" neprázdná.

Rozšíření cesty

Rozšíření cesty vypadá tak, že pro vrchol v ve vzdálenosti $d(v)$ zkusíme pro každou hranu $\{v, w\}$, zda

$$d(w) > d(v) + delka(\{v, w\}).$$

Pokud ano, $d(w) := d(v) + delka(\{v, w\})$ a oprav pozici vrcholu w ve "frontě".

Analýza

- Algoritmus je popsán a dokázán v mnoha knihách (a skriptech), kupř. Kapitoly z diskrétní matematiky nebo Algebraické algoritmy (Kučera, Nešetřil)...
- Konečnost: V každé iteraci odstraníme z "fronty" jeden vrchol, který se v ní už neobjeví
  (protože mezi vrcholy ve frontě měl nejmenší vzdálenost od startu a vzdálenosti jsou nezáporné).
- Parciální správnosti pomůže invariant:
  V každém kroku evidujeme nejkratší cesty ze startu používající pouze vrcholy již odstraněné z "fronty".
- Z invariantu plyne korektnost.
- Jde jen o modifikovaný algoritmus vlny, tedy hledání do šířky!
- Složitost významně závisí na reprezentaci grafu a na reprezentaci "fronty"!

Poznámky

- Pokud jde o graf neohodnocený (délka všech hran je 1), potom se z Dijkstrova algoritmu stane obyčejný algoritmus vlny.
- Aplikace grafových algoritmů: Zde začíná teoretická informatika. :-)
- Příklady využití grafových algoritmů:
- Thesseus a Minotaurus,
- Král (či jiné figurky) na šachovnicích různých tvarů s různě pozakazovanými políčky,
- ...

Medián v lineárním čase

- Kdysi byl algoritmus Quicksort.
- Problémem bylo, jak volit pivot.
- Volíme-li špatně, děláme mnoho iterací rekurze.
- Hledáme-li pivot dlouho, nepříjemně to trvá.
- Jak hledat medián v lineárním čase?
- Ve skutečnosti nebudeme hledat medián, ale k-tý nejmenší prvek.

Medián v lineárním čase

- Rozděl vstup na pětice,
- v každé pětici najdi medián,
- najdi medián mediánů (tedy medián mezi mediány pětic),
- rozděl vstup na menší a větší,
- zjisti, zda se k-tý nejmenší nachází mezi většími nebo menšími
- pokračuj s hledáním příslušné hromádce.

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).
- Jak "pokračovat na příslušné hromádce?"
- Rekurzívně (zavolej se buďto na menší hromádku a hledej k-tý nejmenší, nebo máme-li hledat v hromádce větších hodnot budiž l počet prvků na menší hromádce a hledej v hromádce větších $k - l$-tý nejmenší.

Proč je algoritmus lineární?

Protože:

- mediánů pětic je přibližně pětina délky vstupu,
- hromádka menších čísel stejně jako hromádka větších čísel bude mít velikost aspoň 3/10,
- tudíž každá z hromádek bude mít též velikost nejvýš 7/10.
- Zbytek je jen indukce.

Indukce:

Složitost algoritmu vede k rekurenci:

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + kn.$$

Ukážeme, že existuje l takové, že $T(n) \leq ln$:

$$T(n) \leq \frac{ln}{5} + \frac{7ln}{10} + kn = kn + \frac{9}{10}ln$$

a tedy stačí volit $l \geq 10k$.

Odstranění rekurze obecně

- Rekurze je pěkná věc, ale podle teorie lze každý program napsat v podobě jednoho jediného cyklu (while) bez volání dalších funkcí.
- Sice takový program není k přečtení, ale jedná se o zajímavý teoretický závěr.
- Jak to udělat?
- Uděláme totéž, co za nás udělá překladač, když spustíme rekurzi, tedy...

Odstranění rekurze

- Vyrobíme nové lokální proměnné, skočíme na správné místo programu a zapamatujeme si, kam se máme vrátit.
- Ve skutečnosti si jen (na zásobník) uložíme údaje o dosavadních datech, nahradíme daty "zarekurzenými"a údaj odkud jsme se "zavolali"a skočíme na "začátek"funkce.
- Při "odrekurzení"poznamenáme návratové údaje, vytáhneme ze zásobníku původní obsahy lokálních proměnných a údaj odkud jsme se zavolali (a skočíme tam).

Quicksort s logaritmickou pamětí

- Quicksort může vytvořit velkou a malou část.
- Pak potřebujeme lineárně paměti při klasickém rekurzívním řešení.
- Při třídění druhé části nepotřebujeme návratové údaje.
- Proto napřed setřídíme část, která je menší (co do množství dat).
- A problém můžeme řešit hybridně, tedy na první část se rekurzívně zavolat (i když i v tomto případě umíme rekurzi odbourat, neumíme odbourat její paměťové nároky).
- Tím, že rekurzi (nebo její náhražku) spustíme jen na menší část, pracujeme v i-té úrovni rekurze nejvýše s $\frac{n}{2^i}$ hodnotami.

Quicksort s omezenou rekurzí

Hybridní implementace v pseudokódu

```
procedure quicksort(levy,pravy);
begin
      while (levy<>pravy) do
      begin index_pivota:=rozdel;
            if(index_pivota>(pravy-levy)/2) then
            begin quicksort(index_pivota+1,pravy);
                  pravy:=levy+index_pivota
            end else begin
                  quicksort(levy,index_pivota);
                  levy:=levy+index_pivota+1;
end;  end;  end;
```

Textové a binární soubory

Zhodnocení situace

- V zimě jsme se učili pracovat s textovými soubory.
- Proměnnou typu `text` jsme napojili na soubor, otevřeli jsme, četli, zapisovali a zavřeli.
- Široká použitelnost, občas ale chceme naprogramovat například databázi (knih v knihovně).
- K tomu by se hodil soubor úplně jiných vlastností, ve kterém půjde lépe vyhledávat.
- K tomu přesně lze použít binární soubory.
- Jedná se o soubor sestávající z prvků popsané struktury, které můžeme číst resp. zapisovat.
- Jelikož známe velikost dotyčné struktury, můžeme i vyhledávat.

Technické prostředky

- Většinou se shodují s textovými soubory, nicméně:
- Uděláme proměnnou typu `file of nacitany_typ`.
  Například: `var f:file of nesmysl;`
- Funkce `assign`, `reset`, `rewrite`, `read`, `write` a `close` fungují úplně stejně (tedy jako první argument jim předáme příslušnou proměnnou typu `file`.
- Pozor na `append`!
- Hlavní rozdíl:
  - `filesize` – zjistí počet záznamů v souboru,
  - `seek` – nastaví ukazatel na příslušné místo.

Příklad

telefonní seznam

```
type zaznam=record
       jmeno:string[100];
       cislo:string[20];
end;
var f:file of zaznam;
```

Příklad přidání

do binárního souboru

```
procedure pridej;
var zazn:zaznam;
begin readln(zazn.jmeno);
      readln(zazn.cislo);
      assign(f,'databaze.bin');
      {$I-}
      reset(f);
      {$I+}
      if IOResult<>0 then
            rewrite(F);
      seek(f,filesize(f));
      write(f,zazn);
      close(f);
end;
```

Výpis souboru

```
procedure vypis;
var i:integer;
    zazn:zaznam;
begin
      assign(f,'databaze.bin');
      reset(f);
      for i:=1 to filesize(f) do
      begin
            read(f,zazn);
            writeln('Jmeno: ',zazn.jmeno,', cislo:
',zazn.cislo);
      end;
      close(f);
end;
```

Mazází v binárním souboru

Funkce `truncate`

- Zbytek binárního souboru můžeme smazat pomocí funkce `truncate`.
- Jako parametr jí předáme proměnnou typu file (napojenou na nějaký soubor).
- Příklad:
  ```
  reset(f);
  truncate(f);
  ```
  smaže dosavadní obsah souboru
- podobně jako by udělalo `rewrite(f)`, ovšem pro neexistující soubor by to nový nezaložilo!
- Jak zrušit jeden záznam?
- Přepsat ho posledním a poslední zrušit.

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
```