Anotace

- Dynamické programování,
- diskrétní simulace.

Problémy, které byly

- Přednášející jde tentokrát 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,
- problém batohu.

Pascalův trojúhelník

- Obsahuje kombinační čísla,
- n-tý řádek konkrétně obsahuje hodnoty $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$,
- při výpočtu využíváme toho, že $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,
- pro jednoduchost chceme spočítat jen číslo $\binom{n}{k}$.

Pascalův trojúhelník rekurzívní řešení

- Získali jsme rekurenci $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$,
- z této snadno sestrojíme rekurzívní program:

```
function kombin(n,k:integer):longint;
begin
      if (k=0) or (k=n) then kombin:=1;
      else  kombin:=kombin(n-1,k-1)+kombin(n-1,k);
end;
begin
      kombin(100,50);
end.
```

Mále stejný problém, pořád počítáme to samé mockrát.
Opět přidáme cache na výsledky.

Pascalův trojúhelník

```
const MAX=100;
var cache:array[0..MAX,0..MAX] of longint;
function kom(n,k:integer):longint;
begin
    if cache[n,k]=0 then
    begin if (k=0) or (k=n) then cache[n,k]:=1
       else cache[n,k]:=kom(n-1,k-1)+kom(n-1,k);
    end;
    kom:=cache[n,k];
end;
var n,k:integer;
begin
    read(n,k);
    writeln(kom(n,k));
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];

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.

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".

Diskrétní simulace

- Simulace je modelování, při kterém sledujeme časový průběh děje.
- Každou chvíli chceme něco simulovat. Jak to udělat?
- Dva typy procesů: Spojité a diskrétní.
- Spojité: Například míchání kapalin.
- Diskrétní: Například nákup v obchodu.
- Spojitá simulace se typicky řeší metodami matematické analýzy.
- Diskrétní simulace se programuje.
- Zjednodušování a zkoumání, co je podstatné.

Počítačová simulace

- Cílem je zjistit, jak se model bude chovat pro zadaná data.
- Diskrétní simulace operuje s konečným počtem jevů, z nichž každý běží spočitatelným způsobem.
- Například: Zákazník přejde z oddělení hraček do restaurace, pokladní spočítá útratu...
- Typický příklad u nás používaný: Auta s pískem.

Auta s pískem

- Chceme odvézt hromadu písku z místa A do místa B.
- Máme několik automobilů se známými parametry:
  - rychlost jízdy jednotlivými úseky (tedy doba průjezdu),
  - nosnost (kolik lze naložit),
  - doba nakládání,
  - ...
- Na cestě je *kritická sekce*, ve které se auta nemohou míjet.
- Semafor pustí jen jedno.
- Otázka: Jak dlouho budeme hromadu písku převážet z místa A na místo B, pokud v 8 hodin ráno přijde 10 dělníků s lopatami na obě místa a na místo A přijedou dvě nákladní auta?

Auta s pískem

- Pokud by auta neinteragovala (nezdržovala se a nemusela na sebe čekat), jde o úlohu typu *jeden kopáč vykope příkop za hodinu, za jak dlouho nakope příkop deset kopáčů.*
- Co ale dělat, pokud jedno auto má vyšší nosnost, musí jet nižší rychlostí, auta nelze nakládat současně a před kritickou sekcí je třeba čekat na semaforu?
- Budeme simulovat jednotlivé události:
  - Auto přijelo (do bodu A, do bodu B, ke kritické sekci, na konec kritické sekce),
  - auto odjíždí (z místa A nebo B).
- Uděláme si graf jak po sobě události následují v závislosti na okolí a odsimulujeme (například ručně – budeme tahat autíčky a znamenat si čas).

Další příklady

- Samoobsluha:
  - Zákazník přijde do obchodu, chvíli bloudí mezi jednotlivými odděleními,
  - pak se postaví do fronty u pokladny, zaplatí a odejde.
  - Cíl: Jak zorganizovat vnitřek obchodu, aby v ní zákazníci zbytečně netrávili dlouhý čas (případně utratili co nejvíce)...
- Výtahy:
  - V budově je několik výtahů,
  - lidé chtějí cestovat z patra do patra.
  - Jak dlouho se bude čekat na výtah máme-li přiměřený popis chování lidí?

Jak to naprogramovat?

- Vůbec není podstatné, jak jednotlivé akce probíhají, ale jen jak se mezi nimi přechází (tedy kdy doběhne jedna akce a nastane další).
- Návaznosti akcí jsou dobře definovány!
- Implementujeme tzv. **kalendář událostí**...

Kalendář událostí

- Kalendář událostí je datová struktura osazená operacemi `extract_min`, `insert` a `delete`.
- Operace dělají toto (v tomto pořadí): vyber událost, která se má odehrát jako první (od současného (simulovaného) okamžiku); naplánuj (budoucí) událost; zruš naplánovanou událost.
- Událost má čas (kdy nastane) a popis účasníků (a popis události).
- Příklad: 11:20, **konec přednášky**, účastníci: přednášející (smaže tabuli), studenti (každý někam odejde).

Kalendář událostí

- Jak to realizovat?
- Spojovým seznamem (uspořádaným nebo neuspořádaným),
- haldou,
- polem (pokud umíme shora omezit maximální délku kalendáře),
- ... (použitá datová struktura ovlivňuje složitost).
- Co s procesy, které čekají?

Stavy procesů

- Dobrá, pro začátek stavy procesů:
- běží – proces počítá, tedy jej právě zpracováváme (rozhodujeme, co s ním). Aktivní je vždy nejvýše jeden proces (pokud nemáme pro simulaci paralelní prostředí).
- naplánován – má v kalendáři událostí zařazenu plánovanou událost; víme, kdy se s ním něco stane.
- čeká – bude aktivován jiným procesem – je dobře definováno jakým, ale nemůžeme ještě říci, kdy se tak stane (například auto čeká před kritickou sekcí, až semafor nastaví zelenou v jeho směru, nebo zákazník čeká ve frontě, až budou vybaveni všichni zákazníci před ním).
- ukončený – proces doběhl – takový proces buďto zrušíme, nebo udržujeme pro účely zjištění údajů o jeho běhu.

Neživé procesy

- Budou všechny procesy k nalezení v kalendáři událostí?
- Co když u některého procesu nevíme, kdy může být naplánován?
- Ten v kalendáři naplánován nebude, nesmíme o něm ale ztratit údaje.
- Čekající procesy musíme vhodným způsobem organizovat (nastrkat do vhodných front, případně jiných datových struktur).

Simulační jádro

- je částí hlavního programu pracující s kalendářem událostí.
- Vyzvedne z kalendáře nejbližší událost a zpracuje.
- Pozor, tato událost nás může přimět vyházet z kalendáře některé další události!
- Pozor, obsloužení události může vyústit v naplánování další události procesu, nebo v jeho deaktivaci (čekání ve frontě). Taktéž můžeme některé procesy aktivovat (probudit).

Simulační program

- Zinicializuje kalendář událostí a pomocné fronty,
- načte příslušná data,
- naplánuje iniciální události,
- spustí jádro,
- vypíše výsledky (typicky čas konce simulace).

Příklad: Auta s pískem

- Zjistíme počet a parametry jednotlivých aut,
- co auto, to proces. Auta předjedou ve stanovený čas (naplánujeme pro ně na správnou dobu událost – asi *přijeď k hromadě*).
- Událost *přijeď k hromadě* zjistí, zda je hromada volná. Pokud ne, zařadí se do fronty. Pokud ano, naplánuje událost *odjezd*.
- Událost *odjezd* naplánuje příjezd auta ke kritické sekci a probudí první auto čekající ve frontě u hromady.

Příklad: Auta s pískem – pokračování

- *příjezd ke kritické sekci* zjistí, zda je kritická sekce volná (na semaforu zelená). Pokud ne, zařadí se do fronty u semaforu. Pokud volno je, naplánuje si *odjezd z kritické sekce*.
- *odjezd z kritické sekce* naplánuje příjezd ke staveništi a probudí auta čekající v protisměru.
- *přijeď ke staveništi* a *odjezd ze staveniště* se chová podobně jako u hromady.

Konec

- Teď už to jen naprogramovat...
- ...děkuji za pozornost...