Anotace

- Dámy na šachovnici – dominance a nezávislost.
- Problémy řešitelné "vyplněním tabulky":
  - počet korektních uzávorkování pomocí n párů závorek,
  - nalezení všech Youngových tabulek,
  - Pascalův trojúhelník,

Dámy na šachovnici

- Jak rozmístit n dam na šachovnici $n \times n$, aby se vzájemně neohrožovaly?
- Jak rozmístit co největší počet dam, aby se navzájem neohrožovaly? (nezávislost)
- Jak rozmístit co nejmenší počet dam, aby ohrožovaly celou šachovnici? (dominance)

n dam na $n \times n$

- Každá dáma přijde do jednoho řádku,
- stačí dámu postupně zkoušet umísťovat na dosud neohrožená pole a spustit rekurzi na další řádky:

```
function dama(radek:integer);
begin  if radek>pocet_radku then vypis
       else  for i:=1 to pocet_radku do
                    if volny_sloupec[i] and
volna_diag1??? and volna_diag2??? then
                    begin
                          obsad_to_vsechno;
                          dama(radek+1);
                          uvolni_to_vsechno;
                    end;
end;
```

Dámy na šachovnici

- Volné řádky můžeme evidovat v poli, abychom nemuseli prohledávat šachovnici.
- Volné diagonály taky, protože
- pro jednu diagonálu je součet konstantní $(1 + 5 = 2 + 4 = 3 + 3 = 4 + 2 = 5 + 1)$,
- pro druhou diagonálu je konstantní rozdíl $(1 - 1 = 2 - 2 = 3 - 3 = ...)$,
- uděláme tedy tři pole booleanů.

Dámy – dominance a nezávislost

- Situace je horší, nevíme předem, kolik jich bude,
- ale máme horní a dolní odhady. Pro dominanci i nezávislost nejvýše n, pro nezávislost stačí zkoušet každou dámu do jiného řádku.
- Kupříkladu napřed vždy zkus dámu nepřidat, pak až přidat. Tak ovšem nestačí jen najít první řešení, je potřeba projít všechny kandidáty (nikdo nezaručuje, že to s nejméně dámami najdeme první).

Dámy – dominance a nezávislost

- Obecně funkční možnost: Zkusíme $1 - -n$ dam přidávat na šachovnici všemi možnými způsoby a testovat, kolik polí ohrozíme, nebo zda ohrozíme jinou dámu.
- Kdo jaké triky vymyslí, takové má.
- Většinou nestačí pamatovat si, zda je pole ohrožené, ale kolikrát, abychom věděli, zda po odebrání "současné" dámy zůstane ohrožené nebo ne!

Dominance a nezávislost – zobecnění

- Jiné šachovnice (tórus, Möbiův list, Kleinova láhev),
- jiné figurky (věž, kůň, Maharadža...),
- Více problémů spočívajících v horší analýze, kolik figurek stačí a kolik jich je naopak aspoň potřeba, lezení přes okraj pole...

Počet korektních uzávorkování

- Jak budeme řešit?
- Pomocí rekurze podle rostoucího počtu přidaných závorek.
- Uděláme funkci, která:
- zkusí přidat otevírací závorku (rekurze),
- zkusí přidat zavírací závorku (rekurze),
- pokud jsou použity všechny závorky, zvyš počet uzávorkování o 1.

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(lev,prav:integer);
begin
     if lev>prav then
          pridej_zavorku(lev,prav+1);
     if paru>lev then
          pridej_zavorku(lev+1,prav);
     if (lev=prav) and (paru=lev) then
          inc(celkem);
end;
```

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(lev,prav:integer);
begin
     if lev>prav then
          pridej_zavorku(lev,prav+1);
     if paru>lev then
          pridej_zavorku(lev+1,prav);
     if (lev=prav) and (paru=lev) then
          inc(celkem);
end;
```

Jaký problém má toto řešení?

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(lev,prav:integer);
begin
     if lev>prav then
          pridej_zavorku(lev,prav+1);
     if paru>lev then
          pridej_zavorku(lev+1,prav);
     if (lev=prav) and (paru=lev) then
          inc(celkem);
end;
```

Jaký problém má toto řešení?

Počítáme pořád to samé!

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[lev,prav]` =0, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[lev,prav]` =0, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.
- Je-li `cache[lev,prav]`<>0, připočteme tolik platných uzávorkování.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[lev,prav]` =0, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.
- Je-li `cache[lev,prav]`<>0, připočteme tolik platných uzávorkování.
- Rozdíl mezi variantou "rekurze"a "cachovaná rekurze"je rozdíl mezi nepoužitelným a dobrým algoritmem!

Nalezení všech Youngových tabulek

- Youngova tabulka je tabulka sestávající z řádků postupně nerostoucí délky.
- Samotný řádek je Youngova tabulka.
- Jeden sloupec taktéž.
- "Normální" tabulka $m \times n$ také.
- Nesmí se jen objevit širší řádek za užším.
- Pro zadané n vypište všechny Youngovy tabulky s n políčky.
- Nejde o nic jiného, než zjistit, kolika způsoby lze rozložit číslo na sčítance v nerostoucím pořadí.
- Zajímá nás jen jejich počet, nechceme tabulky vypisovat!

Youngovy tabulku – k řešení

- Jak úlohu vyřešit?
- Jako obvykle rekurzí. Budeme si pamatovat, kolik okének ještě zbývá a kolik jich nejvýše smíme dát do jednoho řádku.
  A zkusíme všechno od maxima, až k jedné.

Rekurzívní funkce:

```
procedure pridej_radek(kolik,maximum:integer);
var i:integer;
begin
     if kolik:=0 then inc(pocet)
     else
          for i:=maximum downto 1 do
               pridej_radek(kolik-i,i);
end;
```

Jaký je problém?

Pořád ten samý

(počítáme pořád to samé).

Youngovy tabulky

- Jak z pasti tentokrát?

Youngovy tabulky

- Jak z pasti tentokrát?
- Stejně jako u závorkování:

Youngovy tabulky

- Jak z pasti tentokrát?
- Stejně jako u závorkování:
- Uděláme dvourozměrné pole cache a budeme si do něj značit, kolika způsoby lze rozložit KOLIK, je-li povolená šířka řádku nejvýše MAXIMUM:

```
procedure rozloz(kolik,maximum:integer);
var i,nazacatku:integer;
     if cache[kolik,maximum]<>0 then
           rozloz:=cache[kolik,maximum];
     else
     begin  nazacatku:=pocet;
          if kolik= 0 then inc(pocet);
          else  for i:=maximum downto 1 do
               rozloz(kolik-i,i);
          cache[kolik,maximum]:=pocet-nazacatku;
     end;
end;
```

Pascalův trojúhelník

- Obsahuje kombinační čísla,

Pascalův trojúhelník

- Obsahuje kombinační čísla,
- n-tý řádek konkrétně obsahuje hodnoty $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$,

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 $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

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
```