Backtracking:

Kombinatorikk og permutasjoner

Litt kombinatorikk

- Kombinatorikk:
  - Metoder og formler for å telle opp *antall mulige måter* som vi kan gjennomføre steg-for-steg prosesser på
- Eksempler:
  - Hvor mange ulike LOTTO-rekker finnes det?
  - Hvor mange forskjellige “flush” kan vi få utdelt i poker?
  - Hvor mange ulike oppstillinger av dronninger kan vi lage på et sjakkbrett, slik at ingen står i samme rad eller kolonne?
- Kjennskap til grunnleggende formler fra kombinatorikken er nødvendig for å kunne forstå *kompleksiteten* i en del backtrackingproblemer

Multiplikasjonsprinsippet

- En prosess består av å gjøre n (del)valg i rekkefølge
- I hvert delvalg har vi k_i ulike alternativer, $i = 1, 2, \ldots, n$
- *Hele* prosessen kan da utføres på:

$$k_1 \cdot k_2 \cdot k_3 \cdot \ldots \cdot k_{n-1} \cdot k_n$$

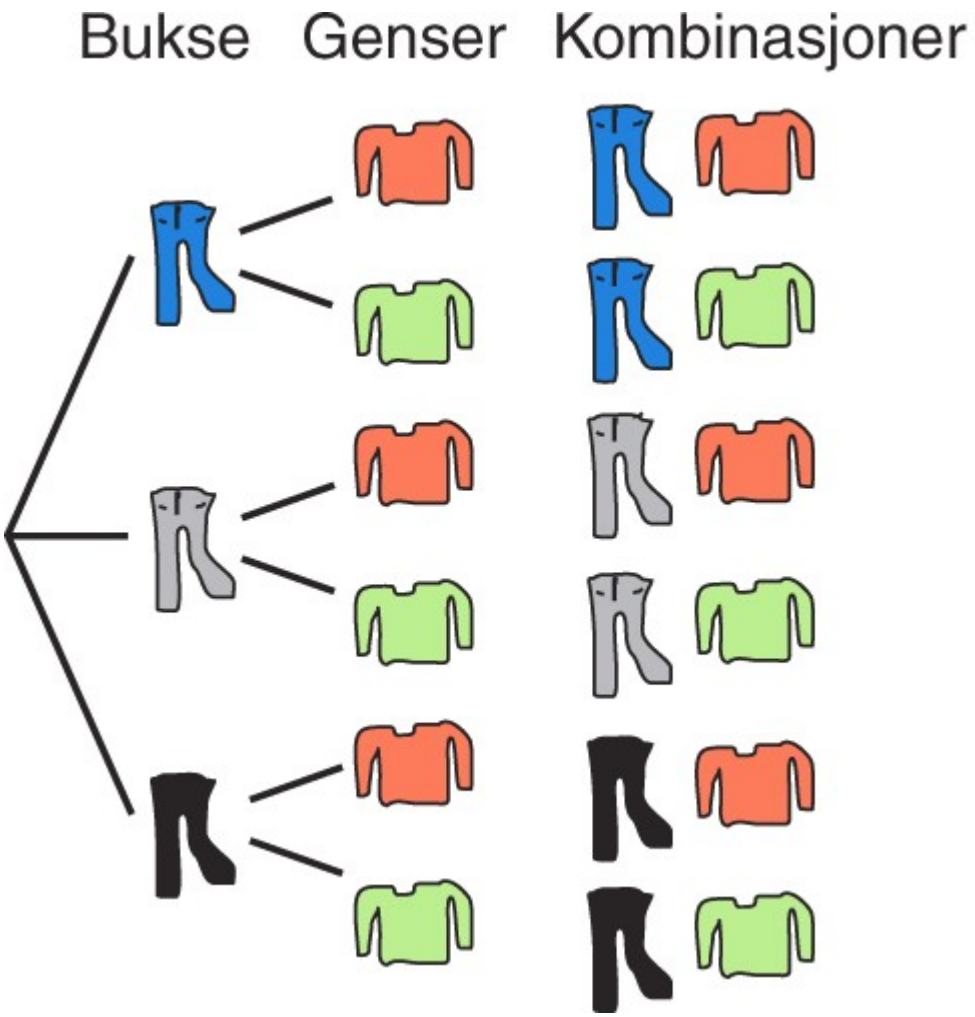
ulike måter

Multiplikasjonsprinsippet: Et eksempel

- Skal velge et antrekk som består av en bukse og en genser
- $n = 2$ delvalg
- $k_1 = 3$ bukser
- $k_2 = 2$ gensere
- Totalt:

$$k_1 \cdot k_2 = 3 \cdot 2 = 6$$

mulige ulike antrekk



Et eksempel til: Svenske bilskilt

- *Vanlige* svenske registreringsnumre for biler består av tre bokstaver etterfulgt av tre siffer (0 – 9)
- Bokstavene I, Q, V, Å, Ä, Ö brukes ikke pga. at de ligner for mye på andre bokstaver/tall
- I alt 23 ulike bokstaver kan velges
- Det er 97 tre-bokstavskombinasjoner som ikke er tillatt å bruke

Antall forskjellige svenske bilskilt

- n = 6 delvalg
- Tre valg av bokstaver, 23 ulike muligheter i hvert valg:

  $k_1 = k_2 = k_3 = 23$

- Tre valg av siffer, 10 ulike muligheter i hvert valg:

  $k_4 = k_5 = k_6 = 10$

- Antall ulike skilt som inneholder ulovlige bokstavkoder:

  $97 \cdot 10 \cdot 10 \cdot 10 = 97\,000$

- Totalt antall forskjellige svenske bilskilt som kan lages:

  $23 \cdot 23 \cdot 23 \cdot 10 \cdot 10 \cdot 10 - 97\,000 = \underline{12\,070\,000}$

Permutasjoner

- Har *n* objekter som alle er ulike
- Objektene skal stilles opp i en eller annen *rekkefølge*
- Det å bestemme en rekkefølge kan sees på som å *nummerere* objektene fra 1 til *n*
- En slik nummerering av *n* objekter kalles for en *permutasjon* av objektene
- Å endre rekkefølger kalles å *permutere* (eller å stokke om, bytte om)
- I mange backtrackingsproblemer vil løsningen være en permutasjon som oppfyller visse krav

Permutasjoner, noen eksempler

- Alle 6 permutasjoner av 123:

  123 132 213 231 312 321

- Alle 24 permutasjoner av ABCD:

  ABCD ABDC ACBD ACDB ADBC ADCB
  BACD BADC BCAD BCDA BDAC BDCA
  CABD CADB CBAD CBDA CDAB CDBA
  DABC DACB DBAC DBCA DCAB DCBA

Hvor mange permutasjoner kan lages?

- Kan bruke multiplikasjonsprinsippet til å regne ut antall mulige *ulike* permutasjoner av *n* objekter
- En permutering er en prosess med n delvalg:

Velg nr. 1	:	n	muligheter
Velg nr. 2	:	$n - 1$	muligheter
.		.	.
.		.	.
.		.	.
Velg nr. n - 1	:	2	muligheter
Velg nr. n	:	1	mulighet

- Total antall mulige permutasjoner av *n* objekter:

$$n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 = n!$$ (*n*-fakultet)

Antall permutasjoner vokser *raskt*

1! = 1

2! = 2

3! = 6

4! = 24

5! = 120

6! = 720

7! = 5040

8! = 40320

9! = 362880

10! = 3628800

11! = 39916800

12! = 479001600

13! = 6227020800

14! = 87178291200

15! = 1307674368000

16! = 20922789888000

17! = 355687428096000

18! = 6402373705728000

19! = 121645100408832000

20! = 2432902008176640000

Hvordan lage og skrive ut alle permutasjoner?

- Ønsker å lage et Java-program som genererer og skriver ut alle permutasjoner av 1, 2, 3, …, *n*
- Permutasjonene skal lages *systematisk* og i stigende rekkefølge
- Kan bruke et slikt program som utgangspunkt for å løse en rekke backtrackingproblemer, inkludert dronningproblemet
- Lager først et program som kan lage alle *sekvenser* av tallene 1,2, 3, …, *n*

Alle sekvenser av *n* = 3 tall

- $3^3 = 27$ ulike sekvenser
- Bare 3! = 6 av disse er permutasjoner:

111 112 113 121 122 **123** 131 **132** 133
211 212 **213** 221 222 223 **231** 232 233
311 **312** 313 **321** 322 323 331 332 333

- Java-program som lager alle sekvenser av 123:

`sekvens_3.java`

$N = 4$: $4^4 = 256$ sekvenser, $4! = 24$ permutasjoner

1111	1112	1113	1114	1121	1122	1123	1124	1131	1132	1133	1134	1141
1142	1143	1144	1211	1212	1213	1214	1221	1222	1223	1224	1231	1232
1233	**1234**	1241	1242	**1243**	1244	1311	1312	1313	1314	1321	1322	1323
1324	1331	1332	1333	1334	1341	**1342**	1343	1344	1411	1412	1413	1414
1421	1422	**1423**	1424	1431	**1432**	1433	1434	1441	1442	1443	1444	2111
2112	2113	2114	2121	2122	2123	2124	2131	2132	2133	**2134**	2141	2142
2143	2144	2211	2212	2213	2214	2221	2222	2223	2224	2231	2232	2233
2234	2241	2242	2243	2244	2311	2312	2313	**2314**	2321	2322	2323	2324
2331	2332	2333	2334	**2341**	2342	2343	2344	2411	2412	**2413**	2414	2421
2422	2423	2424	**2431**	2432	2433	2434	2441	2442	2443	2444	3111	3112
3113	3114	3121	3122	3123	**3124**	3131	3132	3133	3134	3141	**3142**	3143
3144	3211	3212	3213	**3214**	3221	3222	3223	3224	3231	3232	3233	3234
3241	3242	3243	3244	3311	3312	3313	3314	3321	3322	3323	3324	3331
3332	3333	3334	3341	3342	3343	3344	3411	**3412**	3413	3414	**3421**	3422
3423	3424	3431	3432	3433	3434	3441	3442	3443	3444	4111	4112	4113
4114	4121	4122	**4123**	4124	4131	**4132**	4133	4134	4141	4142	4143	4144
4211	4212	**4213**	4214	4221	4222	4223	4224	**4231**	4232	4233	4234	4241
4242	4243	4244	4311	**4312**	4313	4314	**4321**	4322	4323	4324	4331	4332
4333	4334	4341	4342	4343	4344	4411	4412	4413	4414	4421	4422	4423
4424	4431	4432	4433	4434	4441	4442	4443	4444				

Antall sekvenser av *n* tall

- Multiplikasjonsprinsippet gir antall sekvenser av *n* tall:
  - *n* delvalg, ett for hvert siffer I sekvensen
  - *n* alternativer I hvert delvalg
  - Totalt antall ulike sekvenser:

    $n \cdot n \cdot n \cdot \ldots \cdot n = n^n$

- Vanskelig (umulig?) å skrive et generalt **iterativt** program for å lage alle sekvenser for vilkårlig *n*:
  - Java-program som lager alle sekvenser av 12345:

    `sekvens_5.java`

Rekursivt program for å lage og skrive ut alle sekvenser av tallene 1, 2, …, n

- Sekvensene bygges opp systematisk og **rekursivt**, ved at vi etter tur setter alle tallene inn som nummer en, deretter som nummer to osv.
- Bruker en *global* heltallsarray **p** av lengde *n* til å lagre alle sekvensene etterhvert som de genereres
- Rekursjonen følger indeksene i **p**

```
void lagSek(int indeks)
{
   if (indeks == n)
      skrivSek();
   else
   {
      for (int i = 0; i < n; i++)
      {
          p[indeks] = i + 1;
          lagSek(indeks + 1);
      }
   }
}
```

Java-kode med testprogram: `sekvens.java`

Program for å lage og skrive ut alle permutasjoner av tallene 1, 2, …, *n*

- Permutasjonene bygges opp systematisk og rekursivt, ved at vi etter tur setter alle tallene inn som nummer en, deretter som nummer to osv.
- Bruker en *global* heltallsarray **p** av lengde *n* til å lagre permutasjonene som genereres
- Rekursjonen følger indeksene i **p**
- Tar vare på hvilke av tallene 1, 2, .., *n* som allerede er brukt i en permutasjon, i en global boolsk array **brukt**, (initielt lik *false*) slik at vi ikke setter inn samme tall to ganger

```
void lagPerm(int indeks)
{
   if (indeks == n)
      skrivPerm();
   else
   {
      for (int i = 0; i < n; i++)
      {
         if (!brukt[i])
         {
            p[indeks] = i + 1;
            brukt[i] = true;
            lagPerm(indeks + 1);
            brukt[i] = false;
         }
      }
   }
}
```

Java-kode med testprogram: permutasjon.java