UNIVERSITETET
I OSLO

INF2220: Forelesning 7

Kombinatorisk søking

Oversikt

- Rekursjon - oppsummering
- Generering av permutasjoner
  - Lett: Sekvens-generering
  - Vanskelig: Alle tallene må være forskjellige
- Eksempel: Finne korteste reiserute
- Mer direkte generering av permutasjoner

Les notatet ”Kombinatorisk søking, rekursjon, avskjæring” av Stein Krogdahl og Arne Maus.

Rekursjon

En rekursiv metode er en metode som kaller seg selv.

Huskeregler:

- Det må alltid finnes et basistilfelle som kan løses uten rekursjon.
- De rekursive kallene må gå i retning av et basistilfelle.
- Design-regel: Anta at de rekursive kallene fungerer.
- Ikke løs samme instans av et problem i separate rekursive kall!

Oppgave

For hver av de to metodene under:

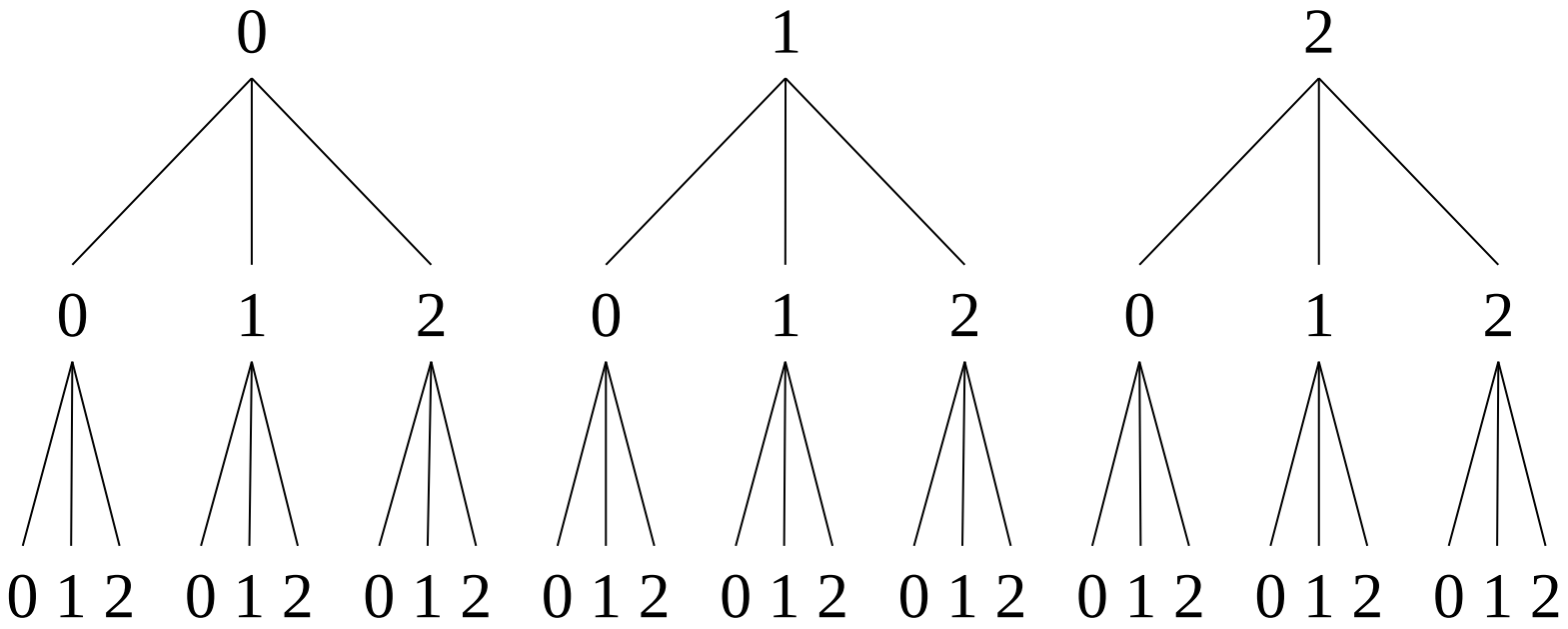
- Hva er basistilfellet?
- Hvordan går de rekursive kallene i retning av basistilfellet?

```
void f1(int n) {
    System.out.println(n);
    if (n > 1) {
        f1(n - 1);
    }
    System.out.println(n);
}
```

```
void f2(int n) {
    if (n > 0) {
        System.out.print("*");
        f2(- (n – 1));
    } else if (n < 0) {
        System.out.print("!");
        f2(- (n + 1));
    }
}
```

Sekvens-generering

Alle mulige sekvenser av lengde tre av tallene 0, 1 og 2:



Sekvens-generering: mulig implementasjon

```
// Startkall: gen3(0);

int[] p = new int[3];

void gen3(int plass) {
    for (int siffer = 0; siffer < 3; siffer++) {
        p[plass] = siffer;
        if (plass < 2) {
            gen3(plass + 1);
        } else {
            System.out.println(""+p[0]+p[1]+p[2]);
        }
    }
}
```

Generalisering

UNIVERSITETET I OSLO

Alle mulige sekvenser av lengde n av tallene fra 0 til n-1:

```
class Gen {
    int[] p; int n;

    Gen(int i) { n = i; p = new int[n]; }

    void gen(int plass) {
        for (int siffer = 0; siffer < n; siffer++) {
            p[plass] = siffer;
            if (plass < n - 1) {
                gen(plass + 1);
            } else {
                for (int i = 0; i < n; i++) {
                    System.out.print(p[i]);
                }
                System.out.print("\n");
            }
        }
    }
}
```

Kombinatorisk søking

Har: Et endelig antall elementer.
Skal: Finne en rekkefølge, gjøre et utplukk, lage en oppdeling, …

Eksempler:

- Plassere 8 dronninger på et sjakkbrett slik at ingen av dronningene kan slå hverandre.
- Finne korteste rundtur blant 'n' byer hvor hver by bare besøkes en gang (avstanden mellom ethvert par av byer er kjent).

Vi har da problemer hvor vi må:

1. Lage alle interessante kombinasjoner.
2. Teste om en kombinasjon er en løsning på det aktuelle problemet.

NB: Ofte slår vi disse sammen, slik at vi bare lager kombinasjoner som er potensielle løsninger.

Permutasjoner

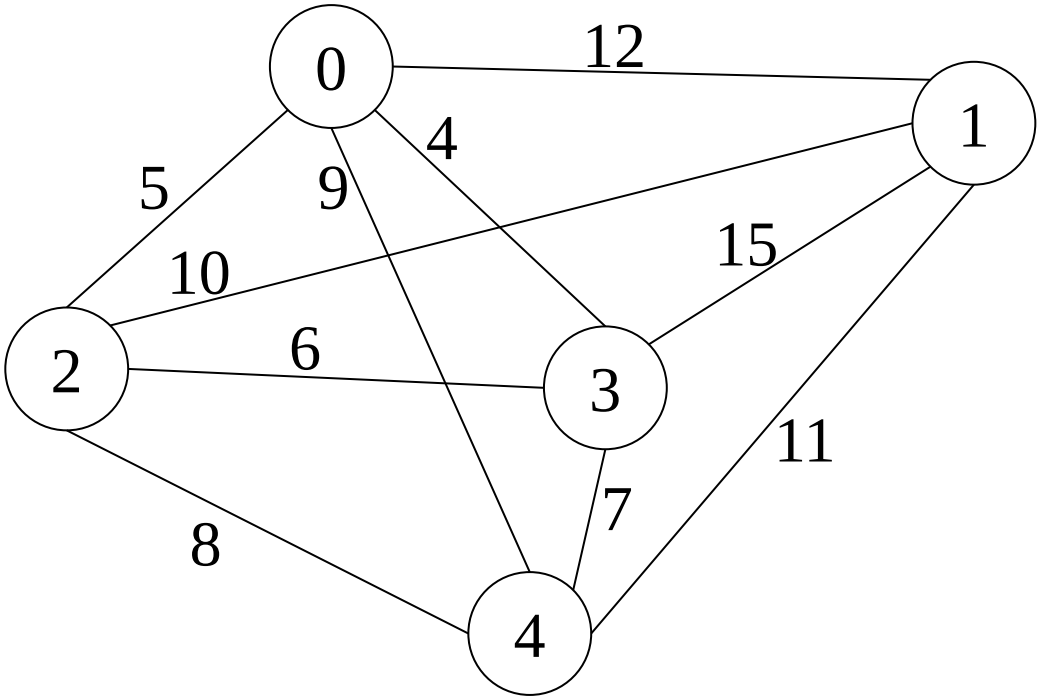
UNIVERSITETET I OSLO

Alle mulige permutasjoner av tallene fra 0 til n-1:

```
class Gen {
    int[] p; int n; boolean[] brukt;
    Gen(int i) {
        n = i; p = new int[n]; brukt = new boolean[n];
        for (int j=0; j<n; j++) { brukt[j] = false; }
    }

    void gen(int plass) {
        for (int siffer = 0; siffer < n; siffer++) {
            if (!brukt[siffer]) {
                brukt[siffer] = true;
                p[plass] = siffer;
                if (plass < n - 1) { gen(plass + 1); }
                else { < Lever p til videre bruk. > }
                brukt[siffer] = false;
            }
        }
    }
}
```

Eksempel: Korteste reiserute



Korteste vei: 36.0

Korteste reiserute (by nr): 0 2 1 4 3 0

Korteste reiserute

UNIVERSITETET I OSLO

- nytt navn finnVeier, variabelnavn ”by” i stedet for ”siffer”
- startkall: finnVeier(1) – p[0] = 0 er startbyen

```
void finnVeier(int plass) {
    for (int by = 1; by < n; by++) {
        if (!brukt[by]&&
            !(lengde + avstand[p[plass-1]][by] >= minLengde)){
            brukt[by] = true;
            lengde += avstand[p[plass-1]][by];
            p[plass] = by;
            if (plass < n - 1) {
                finnVeier(plass + 1);
            } else if (lengde + avstand[by][0] < minLengde) {
                minLengde = lengde + avstand[by][0];
                < Lever p til videre bruk. >
            }
            brukt[by] = false;
            lengde -= avstand[p[plass-1]][by];
        }
    }
}
```

float lengde: lengden på nåværende rute

float[][] avstand: avstanden mellom ethvert par av byer

float minlengde: korteste hittil

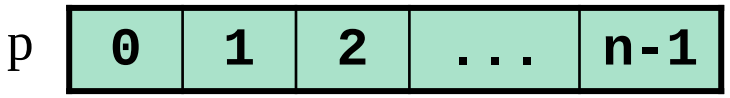
Alternativ generering av permutasjoner

En mer direkte (og mer effektiv?) generering av permutasjoner.

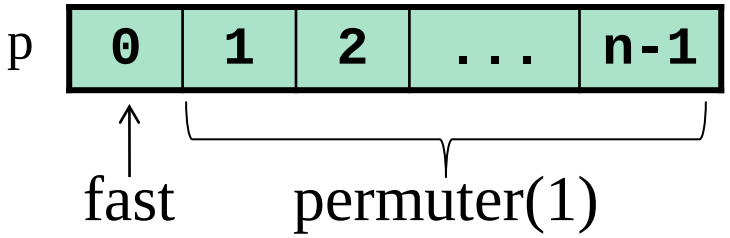
- Initielt: Arrayen p inneholder de aktuelle elementene.

- Metoden permuter(i):
  - Skal generere alle permutasjoner fra og med indeks i.
  - Skal levere tilbake arrayen p slik den var da metoden ble kalt.

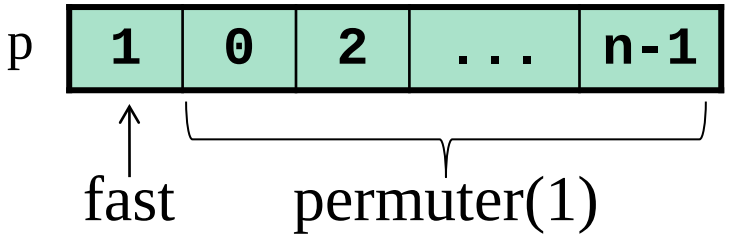
Initielt:



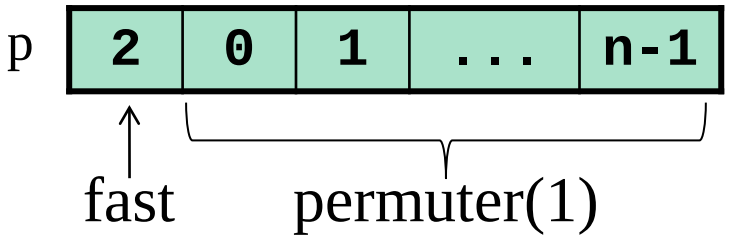
Alle som begynner med 0:



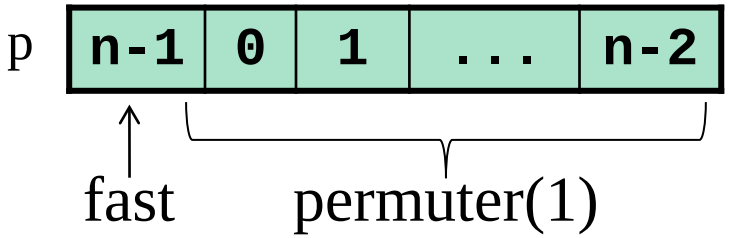
Alle som begynner med 1:



Alle som begynner med 2:



Alle som begynner med n-1:



Direkte generering av permutasjoner

```
// Startkall: permuter(0);

void permuter(int i) {
    if (i == n – 1) {
        < Lever p til videre bruk. >
    } else {
        permuter(i + 1);
        for (int k = i + 1; k < n; k++) {
            bytt(i,k);
            permuter(i + 1);
        }
        roterVenstre(i);
    }
}
```

Hjelpemetodene bytt og roterVenstre

```
void bytt(i,j) {
    int tmp = p[i];
    p[i] = p[j];
    p[j] = tmp;
}
```

```
void roterVenstre(int i) {
    int tmp = p[i];
    for (int k = i + 1; k < n; k++) {
        p[k – 1] = p[k];
    }
    p[n-1] = tmp;
}
```

Repetisjon kombinatorisk søking

Har: Et endelig antall elementer.

Skal: Finne en rekkefølge, gjøre et utplukk, …

- Del kravet i en enkel og en vrien del.
- Test mest mulig av det vriene kravet underveis.
- Kjenn igjen håpløse delløsninger så tidlig som mulig.