Perrinkimas ir grįžimo metodas

Modern computers are so fast that brute force can be an effective and honourable way to solve problems.
Šiuolaikiniai kompiuteriai yra tokie spartūs, kad perrinkimas gali būti efektyvus ir garbingas būdas uždaviniams spręsti.
Steven S. Skienna, Miguel A. Revilla, „Programming Challenges“

1852 metais matematikas F. Gatris (Francis Guthrie) paskelbė hipotezę, teigiančią, jog kiekvienam žemėlapiui nuspalvinti taip, kad jokios dvi gretimos valstybės nebūtų nuspalvintos ta pačia spalva, pakanka keturių spalvų. Daugelis matematikų siūlė šios hipotezės įrodymus, tačiau vis išaiškėdavo, kad jie neteisingi [1]. Hipotezė pagaliau tapo teorema (buvo įrodyta) 1976 metais, o dalis įrodymo rėmėsi kompiuteriu išnagrinėtomis 1476 situacijomis. Kompiuterio programa veikė šimtus valandų, o žmonėms nepakako ir šimto metų. Taigi nors perrinkimo metodai dažnai būna neefektyvūs, spartėjant kompiuteriams šis sprendimo būdas (visų galimų sprendinių išbandymas) tam tikrais atvejais gali būti priimtinas, ypač jei perrinkimą pavyksta optimizuoti.

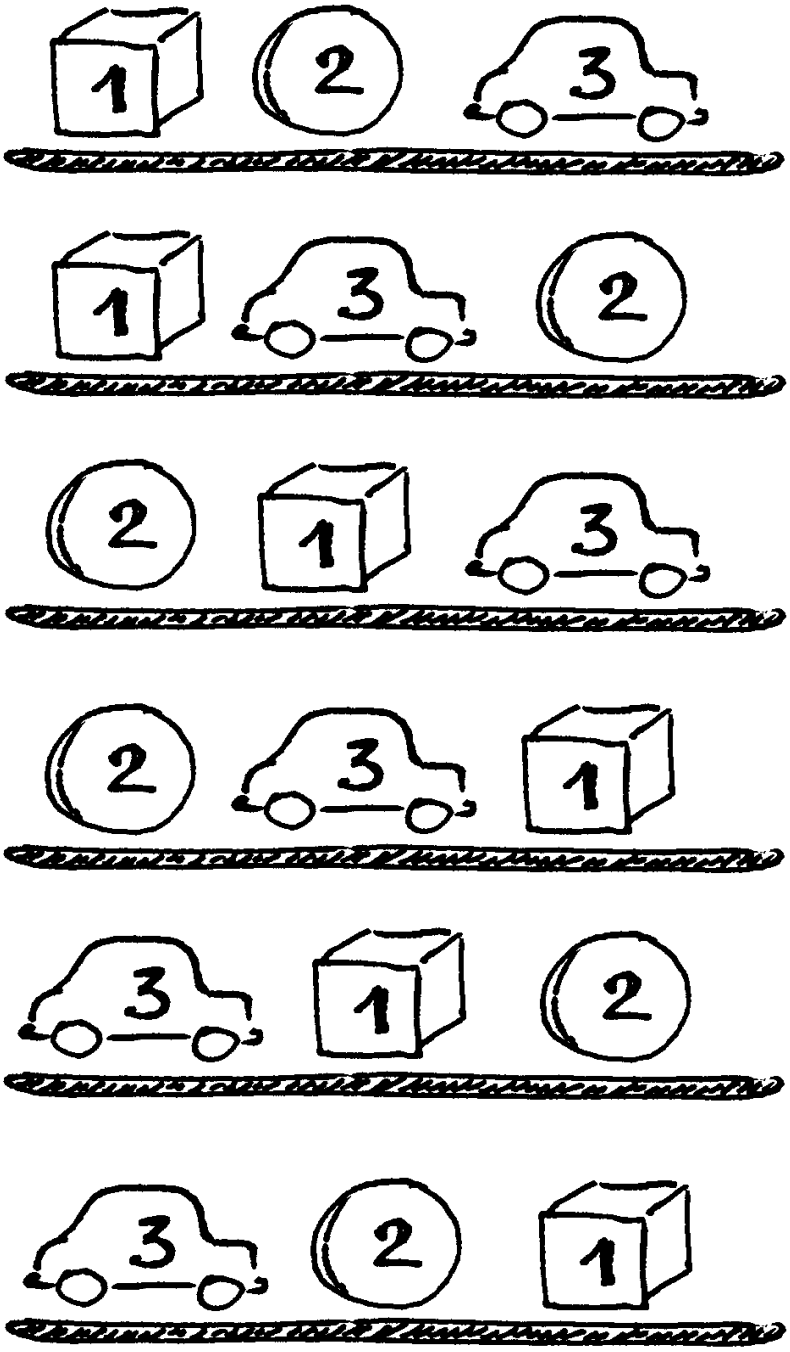
Formaliai **perrinkimą** galima apibrėžti kaip uždavinių sprendimo metodą, kai išbandomi visi galimi sprendiniai.

Šiame skyrelyje susipažinsime su patogiu metodu perrinkimui realizuoti – grįžimo metodu. **Grįžimo metodas** (angl. *Backtracking*) – tai sistemingas būdas spręsti uždaviniams, kurių sprendinys yra kintamųjų $p_1, p_2, \ldots, p_n$ reikšmių rinkinys, tenkinantis kokius nors reikalavimus. Prisiminkime, pavyzdžiui, *Keliaujančio pirklio uždavinį* [2]: šiuo atveju kintamųjų $p_1, p_2, \ldots, p_n$ reikšmėms reikia priskirti skirtingus miestų numerius taip, kad ši miestų aplankymo tvarka ir būtų pageidaujamas maršrutas.

Pagrindinė grįžimo metodo idėja tokia: paeiliui renkamos visų galimų kintamųjų reikšmės ir tikrinama, ar tenkinami reikalavimai, o radus sprendinį arba situaciją, kai reikalavimai netenkinami, grįžtama per vieną žingsnį atgal ir parenkama nauja atitinkamo kintamojo reikšmė.

Programuojant grįžimo metodą dažnai naudojama rekursija. Panagrinėsime abstrakčių kombinatorinių uždavinių sprendimų schemas bei porą konkrečių uždavinių.

Kėlinių generavimas



1 pav. Visi galimi trijų prekių išdėstymo lentynoje būdai

Sakykime, parduotuvės lentynoje vienoje eilėje reikia išdėlioti n skirtingų prekių. Raskime visus skirtingus būdus, kaip tai padaryti. Uždavinys yra ekvivalentus visų n ilgio kėlinių be pasikartojimų generavimo uždaviniui.

Kas gi tas kėlinys be pasikartojimų? Tarkime, turime aibę iš n elementų. Kiekviena visų (skirtingų) n elementų seka vadinama kėliniu be pasikartojimų. Taigi kėliniai vienas nuo kito skiriasi tik elementų išsidėstymu vienas kito atžvilgiu.

Parašykime algoritmą, kuris išspausdintų visus prekių išdėstymo būdus. Prekes laikysime sunumeruotomis nuo 1 iki n.

Atkreipkite dėmesį – šis uždavinys priskiriamas įžangoje minėtai uždavinių klasei: m-ojoje vietoje padėtą prekę (prekės numerį) pažymėjus p_m, reikia rasti kintamųjų $p_1, \ldots, p_n$ reikšmių (kintančių nuo 1 iki n) rinkinius, tenkinančius vieną reikalavimą – visos reikšmės turi būti skirtingos; tuos rinkinius atspausdinti.

Uždavinį galima išreikšti rekursyviai, t. y. suskaidyti į tokius pat, tik mažesnius, uždavinius. Tegu procedūra generuok(m, n) priskirs reikšmes kintamiesiems nuo p_m-ojo iki p_n-ojo. Tuomet jos veikimas galėtų būti toks:

- Jei $m \leq n$, imti po vieną visas dar lentynoje nepadėtas prekes ir su kiekviena atlikti tokius veiksmus:
  - Prekę padėti į m-ąją poziciją lentynoje (pm := prekės numeris).
  - Sudėti į lentyną likusias prekes (priskirti reikšmes kintamiesiems nuo $p_m + 1$ iki p_n) – toks pat uždavinys, taigi iškviesti generuok(m + 1).
  - Prekę, esančią m-ojoje pozicijoje, paimti nuo lentynos.
- Jei $m > n$, tai ši procedūra iškviesta jau išdėliojus visas prekes lentynoje, todėl atspausdiname kintamųjų $p_1, p_2, \ldots, p_n$ reikšmes.

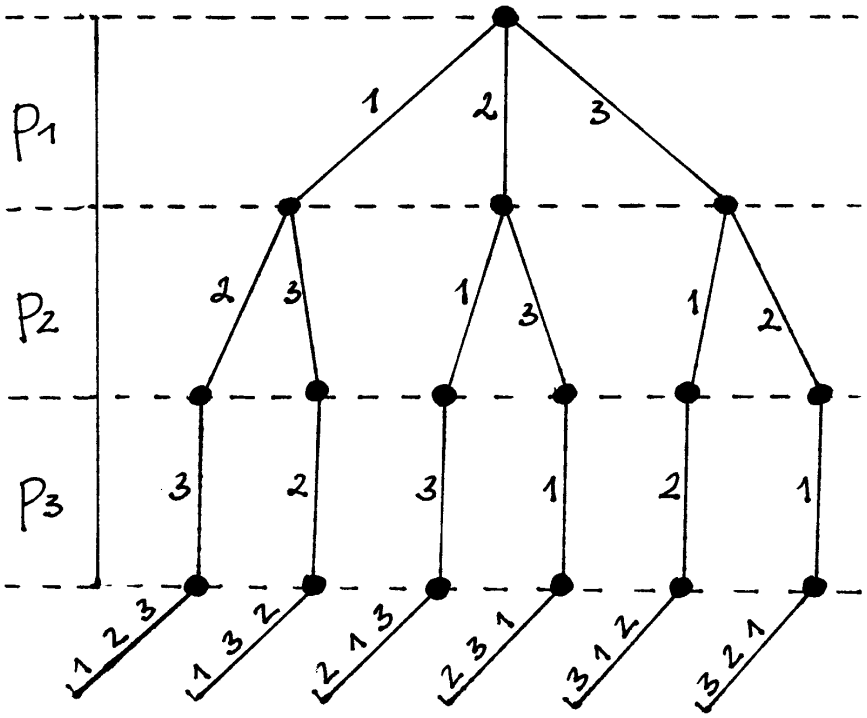
Norint patikrinti, kurios prekės jau sudėtos į lentyną, galima peržiūrėti jau priskirtas reikšmes. Tačiau paprasčiau ir efektyviau paskirti globalų loginį masyvą panaudotas, ir, padėjus į lentyną prekę su numeriu i, pažymėti, jog šis numeris jau panaudotas (panaudotas[i] = true), o paėmus prekę nuo lentynos – atstatyti buvusią reikšmę (panaudotas[i] = false).

```
using namespace std;
int P[200];
bool Panaudotas[200];

void Spausdink(int m);
void Generuok(int m, int n);

int main() {
    int n=5;
```

```
    for(int i=1; i<=n; i++)
    {
        Panaudotas[i]=false;
    }

    Generuok(1,n);

    return 0;
}

void Spausdink(int m)
{
    for(int i=1; i<=m; i++)
    {
        cout << P[i] << " ";
    }
    cout <<     endl;
}

void Generuok(int m, int n)
//m - parenkamas elementas m-tajai pozicijai
{
    if(m>n)

        Spausdink(n);

    else
        for(int i=1; i<=n; i++)
        {
        if(Panaudotas[i]==false)
        {
            Panaudotas[i]=true;
            P[m]=i;
            Generuok(m+1,n);
            Panaudotas[i]=false;
        }
        }
}
```

Kad galėtume išspausdinti visas trijų prekių išdėliojimo lentynoje tvarkas, įvykdome:

```
int n=3;
    for(int i=1; i<=n; i++)
    {
        Panaudotas[i]=false;
    }

    Generuok(1,n);
```



2 pav. Funkcijos generuok vykdymą vaizduojantis medis (n = 3)

Parašytą funkciją nesunku pritaikyti kitiems uždaviniams – vietoj spausdinimo galima atlikti kokius nors kitus veiksmus. Spausdinimą iškėlėme į atskirą funkciją norėdami paryškinti sprendimo struktūrą.

Koks gi parašytos programos sudėtingumas, t. y. kaip atliekamų veiksmų skaičius priklauso nuo n? Algoritmas generuoja visus įmanomus skaičių nuo 1 iki n išdėstymo į eilę būdus. Kiek jų yra? Pirmąjį skaičių galima parinkti n būdų, antrąjį skaičių – $(n - 1)$ būdu (kadangi vienas skaičius jau pasirinktas), trečiąjį skaičių – $(n - 2)$ būdais (du skaičiai jau parinkti) ir t. t. Gauname, kad yra $n(n-1)(n-2)\cdot\dots\cdot 2\cdot 1 = n!$ skirtingų būdų išdėstyti n skaičių į eilę. Taigi procedūros generuok sudėtingumas yra $O(n!)$. Pavyzdžiui, kai $n = 13$, tai vieną atspausdintą eilutę sudaro apie 30 simbolių, o eilučių yra $13! = 6227020800$ ir programa spausdintų daugiau nei 150 gigabaitų teksto... (jei, žinoma, sulauktume veikimo pabaigos).