# Algorytmy i struktury danych

wykład 3

# Plan wykładu:

- Klasy algorytmów – część pierwsza.

# Klasy algorytmów

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj, → Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji

Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

dziel i zwyciężaj → Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji

programowanie dynamiczne → Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

algorytmy zachłanne → Wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepiej rokującego w danej chwili wyboru rozwiązania częściowego.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji

Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

Wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepiej rokującego w danej chwili wyboru rozwiązania częściowego.

Stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

- dziel i zwyciężaj → Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji
- programowanie dynamiczne → Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.
- algorytmy zachłanne → Wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepiej rokującego w danej chwili wyboru rozwiązania częściowego.
- programowanie liniowe → Stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe
- algorytmy siłowe → Polega na rozpatrywaniu wszystkich możliwych rozwiązań i jest wykonywany aż do znalezienia odpowiedniego rozwiązania.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:

- dziel i zwyciężaj → Podział problemu na mniejsze aż do otrzymania problemu łatwego do implementacji
- programowanie dynamiczne → Opiera się na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.
- algorytmy zachłanne → Wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepiej rokującego w danej chwili wyboru rozwiązania częściowego.
- programowanie liniowe → Stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe
- algorytmy siłowe → Polega na rozpatrywaniu wszystkich możliwych rozwiązań i jest wykonywany aż do znalezienia odpowiedniego rozwiązania.
- algorytmy probabilistyczne → Polega na losowym przeszukiwaniu przestrzeni rozwiązań – próby znalezienia rozwiązania są wykonuje się na wg. wskazań pseudolosowych.
- algorytmy heurystyczne
- symulowane wyżarzanie
- algorytmy genetyczne
- algorytmy kwantowe.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych rozwiązań jest najbardziej prawdopodobne

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych rozwiązań jest najbardziej prawdopodobne

Algorytm heurystyczny przeszukujący przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych rozwiązań jest najbardziej prawdopodobne

Algorytm heurystyczny przeszukujący przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu poprzez ewolucję akceptowalnego rozwiązania.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych rozwiązań jest najbardziej prawdopodobne

Algorytm heurystyczny przeszukujący przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu poprzez ewolucję akceptowalnego rozwiązania.

Wykonywane na komputerach kwantowych, operujących na q-bitach oraz zjawisku splątania.

**Klasy algorytmów** – to podział algorytmów ze względu na ich przeznaczenie, na pewne cechy implementacyjne i zakres przetwarzanych danych.

Klasy algorytmów:
dziel i zwyciężaj,
programowanie dynamiczne,
algorytmy zachłanne,
programowanie liniowe,
algorytmy siłowe,
algorytmy probabilistyczne,
algorytmy heurystyczne,
symulowane wyżarzanie,
algorytmy genetyczne,
algorytmy kwantowe.

Ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych rozwiązań jest najbardziej prawdopodobne

Algorytm heurystyczny przeszukujący przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu poprzez ewolucję akceptowalnego rozwiązania.

Wykonywane na komputerach kwantowych, operujących na q-bitach oraz zjawisku splątania.

**Klas i sposobów podziałów algorytmów można wyróżnić więcej.**

# Algorytmy typu „dziel i zwyciężaj”

**Dziel i zwyciężaj** – polega na podziale problemu na mniejsze, aż do otrzymania problemów, których rozwiązanie jest łatwe. Metoda często prowadzi do bardzo efektywnych rozwiązań.

Strategia:

- problem należy podzielić rekurencyjne na dwa lub więcej podproblemów tego samego lub podobnego typu,
- rozwiązania uzyskane po wykonaniu algorytmów częściowych należy następnie scalić.

**Dziel i zwyciężaj** – polega na podziale problemu na mniejsze, aż do otrzymania problemów, których rozwiązanie jest łatwe. Metoda często prowadzi do bardzo efektywnych rozwiązań.

Strategia:

- problem należy podzielić rekurencyjne na dwa lub więcej podproblemów tego samego lub podobnego typu,
- rozwiązania uzyskane po wykonaniu algorytmów częściowych należy następnie scalić.

Etapy pracy algorytmu:

- „dziel” – podział problemu na podproblemy,
- „zwyciężaj” – rekurencyjne rozwiązanie podproblemów,
- „scal” – łączenie rozwiązań z podproblemów.

**Dziel i zwyciężaj** – polega na podziale problemu na mniejsze, aż do otrzymania problemów, których rozwiązanie jest łatwe. Metoda często prowadzi do bardzo efektywnych rozwiązań.

Strategia:
- problem należy podzielić rekurencyjne na dwa lub więcej podproblemów tego samego lub podobnego typu,
- rozwiązania uzyskane po wykonaniu algorytmów częściowych należy następnie scalić.

Etapy pracy algorytmu:
- „dziel” – podział problemu na podproblemy,
- „zwyciężaj” – rekurencyjne rozwiązanie podproblemów,
- „scal” – łączenie rozwiązań z podproblemów.

Uwagi:
- na etapie „zwyciężaj” nie powinny pozostać podproblemy wymagające kolejnych podziałów,
- nie stosuje się tej metody, gdy podział problemu o rozmiarze $n$ powoduje uzyskanie podproblemu o rozmiarze $n$ lub rozmiarze zbliżonym do $n$.

Przykłady algorytmów:
- sortowanie przez scalanie,
- sortowanie szybkie,
- wyszukiwanie binarne.

Zadanie:
Spośród podanych elementów zbioru wybrać najmniejszy i największy.

Rozwiązanie:
- etap „dziel”
- etap „zwyciężaj”

Zadanie:
Spośród podanych elementów zbioru wybrać najmniejszy i największy.

Rozwiązanie:

- etap „dziel” : Podział zbiory na dwa rozłączne podzbiory i wybranie z każdego wartości minimalnej i maksymalnej.
- etap „zwyciężaj”

Zadanie:
Spośród podanych elementów zbioru wybrać najmniejszy i największy.

Rozwiązanie:

- etap „dziel” : Podział zbiory na dwa rozłączne podzbiory i wybranie z każdego wartości minimalnej i maksymalnej.
- etap „zwyciężaj” : Połączenie zbiorów z etapu „dziel” i wybranie wartości minimalnej i maksymalnej.

Zadanie:
Spośród podanych elementów zbioru wybrać najmniejszy i największy.

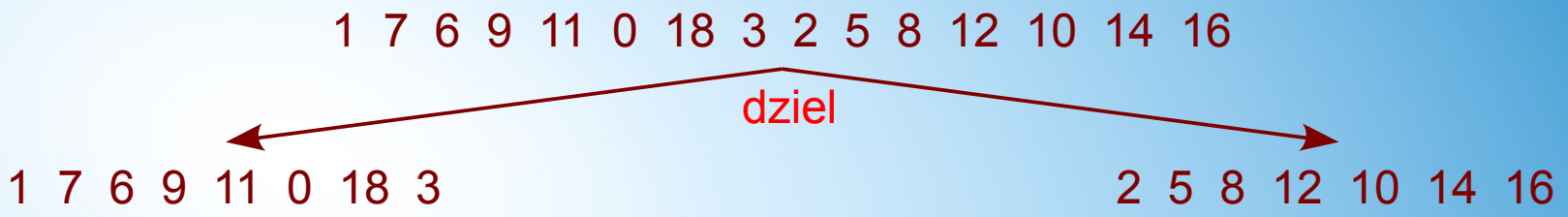
Rozwiązanie:
- etap „dziel” : Podział zbiory na dwa rozłączne podzbiory i wybranie z każdego wartości minimalnej i maksymalnej.
- etap „zwyciężaj” : Połączenie zbiorów z etapu „dziel” i wybranie wartości minimalnej i maksymalnej.

Na etapie „dziel” zbiór wejściowy można podzielić rekurencyjnie na więcej niż dwa podzbiory. Etap „zwyciężaj” będzie wtedy wymagał scalenia wyników częściowych i wybrania wartości minimalnej i maksymalnej ze zbioru powstałego po scaleniu danych częściowych.

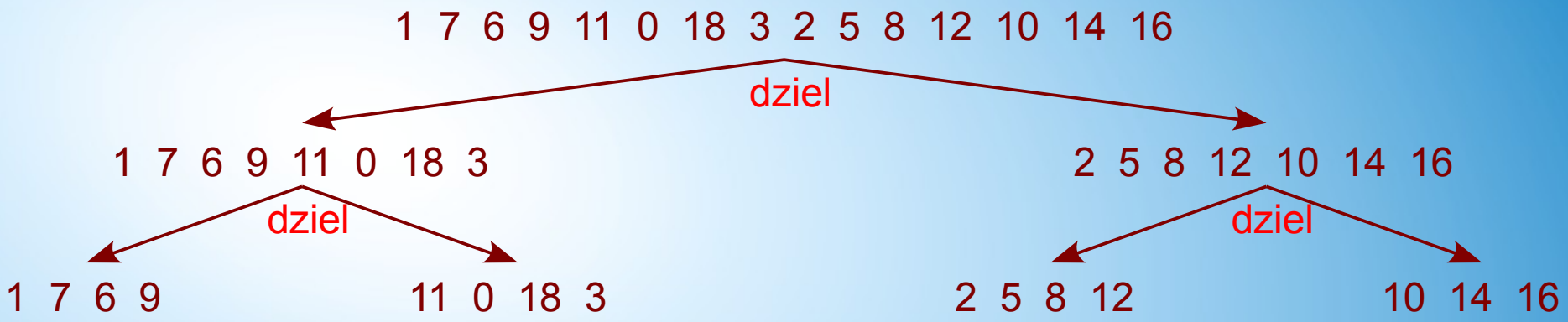
**Dziel i zwyciężaj** – przykład.

1 7 6 9 11 0 18 3 2 5 8 12 10 14 16

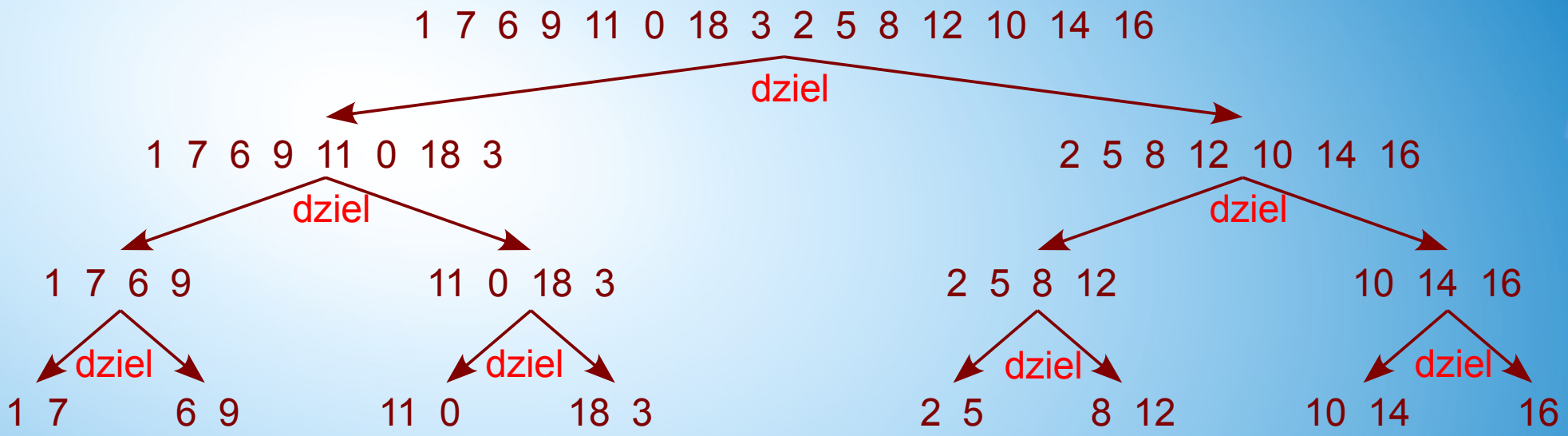
**Dziel i zwyciężaj** – przykład.



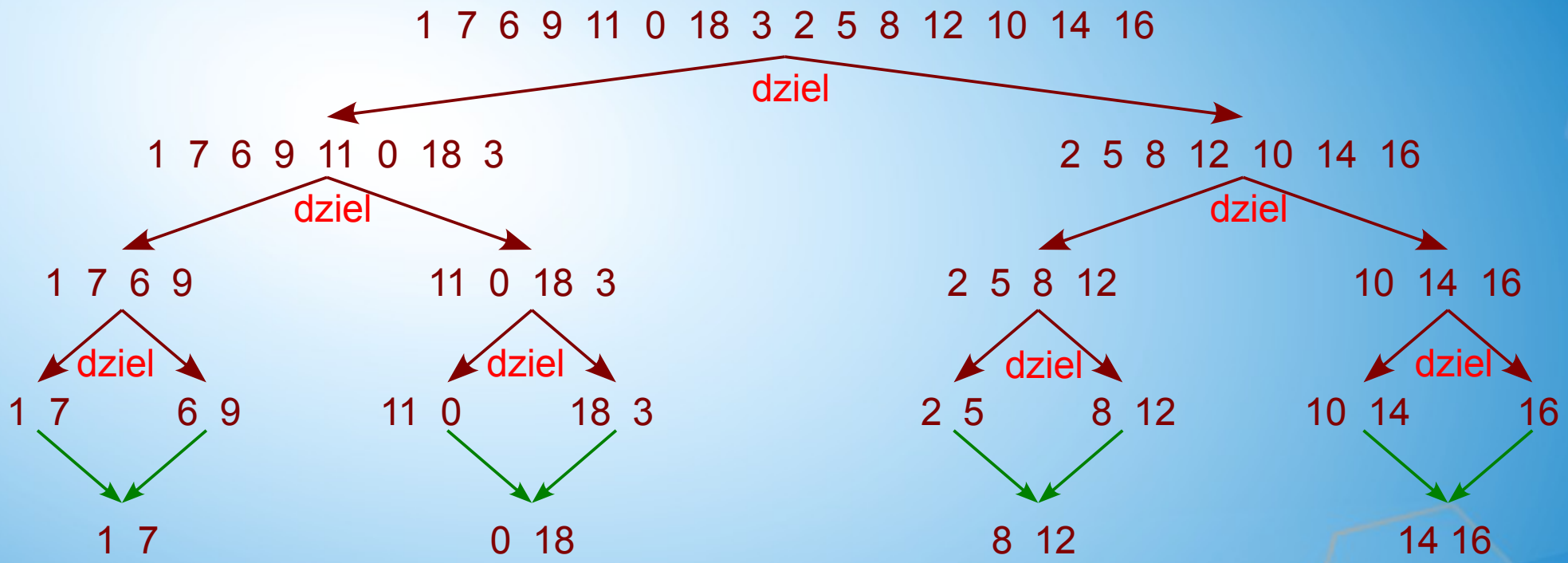
**Dziel i zwyciężaj** – przykład.



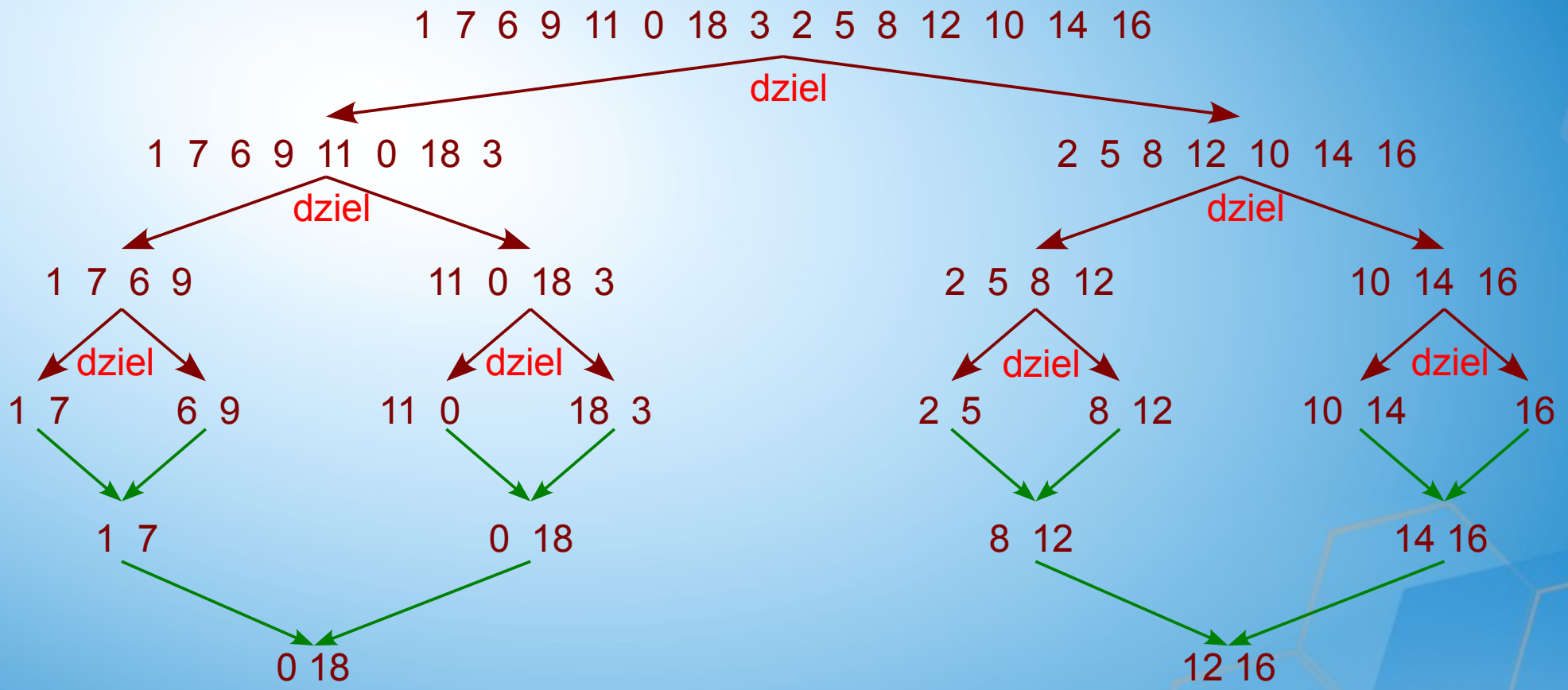
**Dziel i zwyciężaj** – przykład.



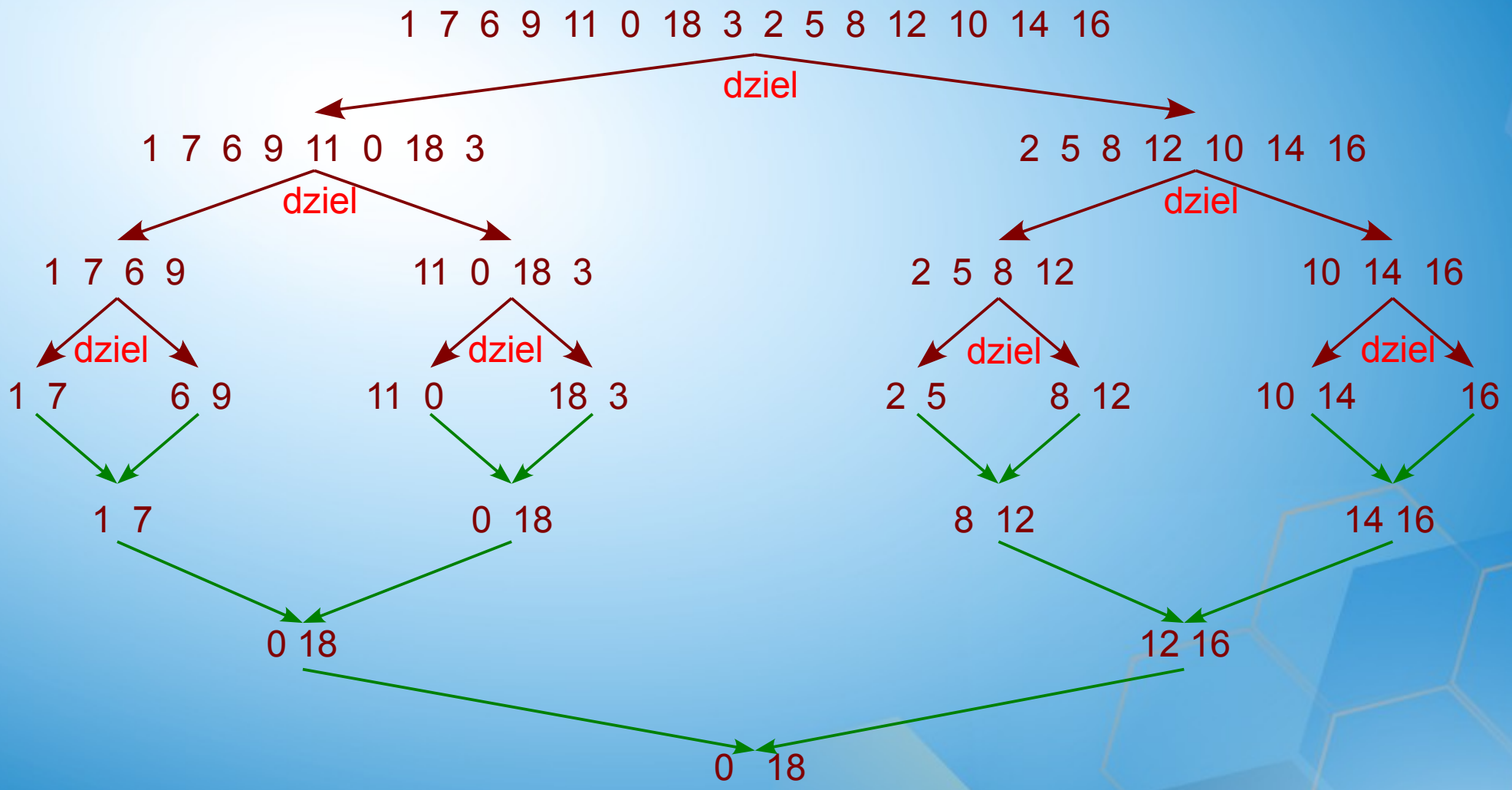
**Dziel i zwyciężaj** – przykład.



**Dziel i zwyciężaj** – przykład.



**Dziel i zwyciężaj** – przykład.



# Programowanie dynamiczne

**Programowanie dynamiczne** – polega na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

Strategia:

- określenie parametrów cechujących problem (parametry i ich liczby),
- określenie funkcji celu będącej funkcją parametrów,
- znalezienie optymalnej funkcji celu poprzez rozwiązanie podproblemów od najprostszego do najbardziej złożonego.

**Programowanie dynamiczne** – polega na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

Strategia:

- określenie parametrów cechujących problem (parametry i ich liczby),
- określenie funkcji celu będącej funkcją parametrów,
- znalezienie optymalnej funkcji celu poprzez rozwiązanie podproblemów od najprostszego do najbardziej złożonego.

Określanie parametrów cechujących problem:

- parametrem może być liczba elementów zbioru danych charakterystycznych dla problemu,
- parametrem może być wartość maksymalna lub minimalna, jaką mogą przyjąć elementy zbioru problemu,
- parametryzowanie problemu może być bardziej złożone.

**Programowanie dynamiczne** – polega na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

Strategia:
- określenie parametrów cechujących problem (parametry i ich liczby),
- określenie funkcji celu będącej funkcją parametrów,
- znalezienie optymalnej funkcji celu poprzez rozwiązanie podproblemów od najprostszego do najbardziej złożonego.

Określanie parametrów cechujących problem:
- parametrem może być liczba elementów zbioru danych charakterystycznych dla problemu,
- parametrem może być wartość maksymalna lub minimalna, jaką mogą przyjąć elementy zbioru problemu,
- parametryzowanie problemu może być bardziej złożone.

Uwagi:
- wraz ze wzrostem liczby parametrów wzrasta zapotrzebowanie algorytmu na pamięć i moc obliczeniową.

**Programowanie dynamiczne** – polega na podziale rozwiązywanego problemu na podproblemy względem kilku parametrów, które cechuje nierozłączność.

Cechy rozwiązania:

- zaprojektowany algorytm ma postać równania rekurencyjnego, opisującego optymalną funkcję celu tego problemu,
- wyznaczenie wartości funkcji celu dla ostatniego podproblemu jest zazwyczaj wynikiem rozwiązania całego zagadnienia,
- metoda jest skuteczna do rozwiązywania problemów NP-trudnych.

Przykłady algorytmów:

- algorytmy znajdujące najdłuższy wspólny podciąg,
- algorytmy rozwiązujące zagadnienie plecakowe,
- obliczanie odległości Levenshteina,
- algorytm Floyda-Warshalla (wyszukiwania najkrótszych ścieżek w grafie ważonym).

Zadanie:
Problem plecakowy – do plecaka należy włożyć jak najwięcej przedmiotów.

Określenie parametrów:
- określenie parametrów: $w_1..w_n$ – wagi przedmiotów, $c_1..c_n$ – wartości przedmiotów,
- celem algorytm jest znalezienie maksymalnej sumy wartości elementów, przy zachowaniu sumy ich wagi mniejszej bądź równej $W$,
- niech $A(i)$ będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej $i$, wtedy $A(W)$ jest rozwiązaniem problemu.

Zadanie:
Problem plecakowy – do plecaka należy włożyć jak najwięcej przedmiotów.

Określenie parametrów:
- określenie parametrów: $w_1..w_n$ – wagi przedmiotów, $c_1..c_n$ – wartości przedmiotów,
- celem algorytm jest znalezienie maksymalnej sumy wartości elementów, przy zachowaniu sumy ich wagi mniejszej bądź równej $W$,
- niech $A(i)$ będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej $i$, wtedy $A(W)$ jest rozwiązaniem problemu.

Funkcja celu A(i):
$A(i)$ jest zdefiniowana rekurencyjnie:
- $A(0) = 0$,
- $A(i) = max \{c_j + A(i - w_j): w_j < i\}$

Zadanie:
Problem plecakowy – do plecaka należy włożyć jak najwięcej przedmiotów.

Określenie parametrów:
- określenie parametrów: $w_1..w_n$ – wagi przedmiotów, $c_1..c_n$ – wartości przedmiotów,
- celem algorytm jest znalezienie maksymalnej sumy wartości elementów, przy zachowaniu sumy ich wagi mniejszej bądź równej $W$,
- niech $A(i)$ będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej $i$, wtedy $A(W)$ jest rozwiązaniem problemu.

Funkcja celu A(i):
$A(i)$ jest zdefiniowana rekurencyjnie:
- $A(0) = 0$,
- $A(i) = max \{c_j + A(i - w_j): w_j < i\}$

Pseudokod rozwiązania:

```
for i := 0 to W do A[i] := 0;
for i := 0 to W do
    for j := 1 to n do
        if w[j] <= i then A[i] := max(A[i], A[i – w[j], c[j]);
```

Zadanie:
Problem plecakowy – do plecaka należy włożyć jak najwięcej przedmiotów.

Określenie parametrów:
- określenie parametrów: $w_1..w_n$ – wagi przedmiotów, $c_1..c_n$ – wartości przedmiotów,
- celem algorytm jest znalezienie maksymalnej sumy wartości elementów, przy zachowaniu sumy ich wagi mniejszej bądź równej $W$,
- niech $A(i)$ będzie największą możliwą wartością, która może być otrzymana przy założeniu wagi mniejszej bądź równej $i$, wtedy $A(W)$ jest rozwiązaniem problemu.

Funkcja celu A(i):
$A(i)$ jest zdefiniowana rekurencyjnie:
- $A(0) = 0$,
- $A(i) = max \{c_j + A(i – w_j): w_j < i\}$

Pseudokod rozwiązania:

```
for i := 0 to W do A[i] := 0;
for i := 0 to W do
   for j := 1 to n do
      if w[j] <= i then A[i] := max(A[i], A[i – w[j], c[j]);
```

Rozwiązaniem dla pustego plecaka jest wartość zero.
Obliczenie wyników dla $A(0)$, $A(1)$... aż do $A(W)$ pozwala określić wynik końcowy.

# Algorytmy zachłanne

**Algorytmy zachłanne** – algorytm wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepszego w danej chwili, wyboru rozwiązania częściowego.

Strategia:

- na danym etapie wyznaczenie możliwych rozwiązań,
- wybranie lokalnie optymalnego rozwiązania dla tego etapu,
- kontynuowanie obliczeń na podstawie ostatnio dokonanego wyboru.

Algorytmy zachłanne cechuje łatwa implementacja. Typowe rozwiązania zachłanne mają charakter optymalizacyjny.

**Algorytmy zachłanne** – algorytm wyznacza rozwiązania w każdym kroku dokonując zachłannego, tj. najlepszego w danej chwili, wyboru rozwiązania częściowego.

Strategia:
- na danym etapie wyznaczenie możliwych rozwiązań,
- wybranie lokalnie optymalnego rozwiązania dla tego etapu,
- kontynuowanie obliczeń na podstawie ostatnio dokonanego wyboru.

Algorytmy zachłanne cechuje łatwa implementacja. Typowe rozwiązania zachłanne mają charakter optymalizacyjny.

Uwagi:
- brak „spoglądania w przyszłość” powoduje, że algorytmy zachłanne mogą nie znajdować optymalnych rozwiązań,
- algorytmy zachłanne często są łączone z innymi klasami algorytmów, wtedy zadaniem części zachłannej jest ograniczenie zbioru możliwych rozwiązań, który będzie dalej optymalizowany przez algorytm dokładny.

Przykłady algorytmów:
- algorytm wyznaczania minimalnego drzewa rozpinającego,
- algorytm odnajdywania najkrótszej ścieżki w grafie (algorytm Dijkstry),
- algorytm szeregowania zadań.

Zadanie:
Jak wydać resztę, mając do dyspozycji monety o nominałach 1, 2 i 5 zł, aby liczba monet była możliwie najmniejsza.

Zadanie:
Jak wydać resztę, mając do dyspozycji monety o nominałach 1, 2 i 5 zł, aby liczba monet była możliwie najmniejsza.

Rozwiązanie:

Problem sprowadza się do określenia zbioru nominałów *N*, jaki należy wydać klientowi, w kolejnych krokach algorytm wykonuje:

1. wybranie nominału *n* do wydania, wartość *n* musi być mniejsze lub równa kwocie do wydania *k*.
2. odjęcie od kwoty *k* nominał *n*: jeśli $k < n$ to powtórzenie kroku 2.
3. jeśli kwota $k > 0$ to idź do kroku 1, jeśli $k = 0$ to KONIEC.

Zadanie:
Jak wydać resztę, mając do dyspozycji monety o nominałach 1, 2 i 5 zł, aby liczba monet była możliwie najmniejsza.

Rozwiązanie:

Problem sprowadza się do określenia zbioru nominałów *N*, jaki należy wydać klientowi, w kolejnych krokach algorytm wykonuje:

1. wybranie nominału *n* do wydania, wartość *n* musi być mniejsze lub równa kwocie do wydania *k*.
2. odjęcie od kwoty *k* nominał *n*: jeśli $k < n$ to powtórzenie kroku 2.
3. jeśli kwota $k > 0$ to idź do kroku 1, jeśli $k = 0$ to KONIEC.

Pseudokod rozwiązania:

```
wybór nominału n;
i := 0;
repeat
    R[i] := n;
    k := k – n;
    wybór nominału n;
until k = 0;
```

# Programowanie liniowe

**Programowanie liniowe** – stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe.

W algorytmach liniowych rozwiązanie polega na minimalizacji lub maksymalizacji funkcji celu.

**Programowanie liniowe** – stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe.

W algorytmach liniowych rozwiązanie polega na minimalizacji lub maksymalizacji funkcji celu.

Strategia:

- określenie warunków ograniczających, mających ogólną postać:
  1. $a_1x_1 + a_2x_2 + \ldots \geq \alpha$
  2. $a_1x_1 + a_2x_2 + \ldots \leq \alpha$
  3. $a_1x_1 + a_2x_2 + \ldots = \alpha$
- maksymalizacja lub minimalizacja funkcji celu, która ma postać:
  $f = \alpha + c_1x_1 + c_2x_2 + \ldots$

**Programowanie liniowe** – stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe.

W algorytmach liniowych rozwiązanie polega na minimalizacji lub maksymalizacji funkcji celu.

Strategia:

- określenie warunków ograniczających, mających ogólną postać:
  1. $a_1x_1 + a_2x_2 + ... \geq \alpha$
  2. $a_1x_1 + a_2x_2 + ... \leq \alpha$
  3. $a_1x_1 + a_2x_2 + ... = \alpha$
- maksymalizacja lub minimalizacja funkcji celu, która ma postać:
  $f = \alpha + c_1x_1 + c_2x_2 + ...$

Uwagi:

- zmienne $x$ są zazwyczaj liczbami rzeczywistymi,
- nie każdy problem liniowy ma rozwiązanie np. ($x_1 \geq 2$, $x_1 \leq 1$),
- żadne z możliwych rozwiązań nie musi być optymalne.

**Programowanie liniowe** – stosuje się do klasy algorytmów, w których ograniczenia i funkcja celu są liniowe.

W algorytmach liniowych rozwiązanie polega na minimalizacji lub maksymalizacji funkcji celu.

Strategia:

- określenie warunków ograniczających, mających ogólną postać:
  1. $a_1x_1 + a_2x_2 + \ldots \geq \alpha$
  2. $a_1x_1 + a_2x_2 + \ldots \leq \alpha$
  3. $a_1x_1 + a_2x_2 + \ldots = \alpha$
- maksymalizacja lub minimalizacja funkcji celu, która ma postać:
  $f = \alpha + c_1x_1 + c_2x_2 + \ldots$

Uwagi:

- zmienne $x$ są zazwyczaj liczbami rzeczywistymi,
- nie każdy problem liniowy ma rozwiązanie np. ($x_1 \geq 2$, $x_1 \leq 1$),
- żadne z możliwych rozwiązań nie musi być optymalne.

Programowanie liniowe ma duże znaczenie w optymalizacji procesów produkcyjnych.

# Algorytmy siłowe

**Algorytmy siłowe** – rozpatrują wszelkich możliwe rozwiązania, aż do znalezienia rozwiązania, które może być zaakceptowane jako wynik.

Strategia:

- wyznaczenie zbioru możliwych rozwiązań,
- sprawdzanie kolejnych rozwiązań ze zbioru wyznaczonego na pierwszym etapie, aż do znalezienia wyniku.

**Algorytmy siłowe** – rozpatrują wszelkich możliwe rozwiązania, aż do znalezienia rozwiązania, które może być zaakceptowane jako wynik.

Strategia:
- wyznaczenie zbioru możliwych rozwiązań,
- sprawdzanie kolejnych rozwiązań ze zbioru wyznaczonego na pierwszym etapie, aż do znalezienia wyniku.

Uwagi:
- algorytmy tego typu nie posiadają żadnej metody ograniczania przestrzeni poszukiwań,
- w przypadku zbioru możliwych rozwiązań o znacznej liczbie elementów, wykonanie algorytmu siłowego może się nigdy nie skończyć,
- dowolny algorytm można zastąpić przez jego odpowiednik siłowy.

**Algorytmy siłowe** – rozpatrują wszelkich możliwe rozwiązania, aż do znalezienia rozwiązania, które może być zaakceptowane jako wynik.

Strategia:
- wyznaczenie zbioru możliwych rozwiązań,
- sprawdzanie kolejnych rozwiązań ze zbioru wyznaczonego na pierwszym etapie, aż do znalezienia wyniku.

Uwagi:
- algorytmy tego typu nie posiadają żadnej metody ograniczania przestrzeni poszukiwań,
- w przypadku zbioru możliwych rozwiązań o znacznej liczbie elementów, wykonanie algorytmu siłowego może się nigdy nie skończyć,
- dowolny algorytm można zastąpić przez jego odpowiednik siłowy.

Możliwe modyfikacje:
- ograniczenie przestrzeni poszukiwań:
  - do pewnych obszarów możliwych rozwiązań
  - w sposób losowy

**Algorytmy siłowe** – rozpatrują wszelkich możliwe rozwiązania, aż do znalezienia rozwiązania, które może być zaakceptowane jako wynik.

Strategia:
- wyznaczenie zbioru możliwych rozwiązań,
- sprawdzanie kolejnych rozwiązań ze zbioru wyznaczonego na pierwszym etapie, aż do znalezienia wyniku.

Uwagi:
- algorytmy tego typu nie posiadają żadnej metody ograniczania przestrzeni poszukiwań,
- w przypadku zbioru możliwych rozwiązań o znacznej liczbie elementów, wykonanie algorytmu siłowego może się nigdy nie skończyć,
- dowolny algorytm można zastąpić przez jego odpowiednik siłowy.

Możliwe modyfikacje:
- ograniczenie przestrzeni poszukiwań:
  - do pewnych obszarów możliwych rozwiązań ⟶ algorytm heurystyczny,
  - w sposób losowy

**Algorytmy siłowe** – rozpatrują wszelkich możliwe rozwiązania, aż do znalezienia rozwiązania, które może być zaakceptowane jako wynik.

Strategia:
- wyznaczenie zbioru możliwych rozwiązań,
- sprawdzanie kolejnych rozwiązań ze zbioru wyznaczonego na pierwszym etapie, aż do znalezienia wyniku.

Uwagi:
- algorytmy tego typu nie posiadają żadnej metody ograniczania przestrzeni poszukiwań,
- w przypadku zbioru możliwych rozwiązań o znacznej liczbie elementów, wykonanie algorytmu siłowego może się nigdy nie skończyć,
- dowolny algorytm można zastąpić przez jego odpowiednik siłowy.

Możliwe modyfikacje:
- ograniczenie przestrzeni poszukiwań:
  - do pewnych obszarów możliwych rozwiązań → algorytm heurystyczny,
  - w sposób losowy → algorytm probabilistyczny.

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa: ???

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion)

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę)

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę),
- liczba wszystkich prób, jakie należy wykonać: $2^k T$, gdzie
  - T – jedna operacja sprawdzenia hasła,
  - k – liczba bitów klucza.

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę),
- liczba wszystkich prób, jakie należy wykonać: $2^k T$, gdzie
  - T – jedna operacja sprawdzenia hasła,
  - k – liczba bitów klucza.
- statystyczna liczba koniecznych prób: $2^{k-1} T$

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę),
- liczba wszystkich prób, jakie należy wykonać: $2^k T$, gdzie
  - T – jedna operacja sprawdzenia hasła,
  - k – liczba bitów klucza.
- statystyczna liczba koniecznych prób: $2^{k-1} T$,
- teoretyczny czas pracy dla systemu o powyższych założeniach wynosi:

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę),
- liczba wszystkich prób, jakie należy wykonać: $2^k T$, gdzie
  - T – jedna operacja sprawdzenia hasła,
  - k – liczba bitów klucza.
- statystyczna liczba koniecznych prób: $2^{k-1} T$,
- teoretyczny czas pracy dla systemu o powyższych założeniach wynosi:

**11 000 000 000 lat**
jedenaście miliardów

Zadanie:
Znaleźć hasło, które zakodowano przy pomocy kodowania AES o kluczu 128 bitów.

Rozwiązanie:
W algorytmie siłowym – sprawdzenie wszystkich możliwych haseł.

Złożoność obliczeniowa:
- liczba komputerów: ~1 000 000 (jeden milion),
- moc obliczeniowa komputerów: ~1 000 000 000 (miliard operacji sprawdzenia haseł na sekundę),
- liczba wszystkich prób, jakie należy wykonać: $2^k T$, gdzie
  - T – jedna operacja sprawdzenia hasła,
  - k – liczba bitów klucza.
- statystyczna liczba koniecznych prób: $2^{k-1} T$,
- teoretyczny czas pracy dla systemu o powyższych założeniach wynosi:

11 000 000 000 lat
jedenaście miliardów

~14 000 000 000 lat
szacowany czas życia Wszechświata

# Algorytmy probabilistyczne

**Algorytmy probabilistyczne** – polega na losowym przeszukiwaniu przestrzeni rozwiązań, przy czym kolejne próby znalezienia rozwiązania są wykonywane na podstawie wskazań pseudolosowych.

Strategia:

1. Dla danej przestrzeni rozwiązań losowanie próbki do sprawdzenia.
2. Sprawdzenie potencjalnego rozwiązania.
3. Jeśli uzyskany wynik nie jest akceptowalny, przejście do etapu 1.

**Algorytmy probabilistyczne** – polega na losowym przeszukiwaniu przestrzeni rozwiązań, przy czym kolejne próby znalezienia rozwiązania są wykonywane na podstawie wskazań pseudolosowych.

Strategia:
1. Dla danej przestrzeni rozwiązań losowanie próbki do sprawdzenia.
2. Sprawdzenie potencjalnego rozwiązania.
3. Jeśli uzyskany wynik nie jest akceptowalny, przejście do etapu 1.

Uwagi:
- algorytmy probabilistyczne nie są deterministyczne – wymagają przypadkowego wskazania potencjalnego rozwiązania,
- algorytmy tego typu są stosunkowo odporne na „dane złośliwe”,
- algorytmy tego są z reguły prostsze i szybsze od ich deterministycznych odpowiedników,
- określono dwa typy algorytmów probabilistycznych:
  - zawsze zwracające poprawne wyniki w bliżej nieokreślonym czasie – Las Vegas,
  - kończące się w ustalonym czasie, ale bez gwarancji ustalenia wyniku – Monte Carlo.

Przykłady algorytmów:
- metoda Monte Carlo (obliczanie pól figur).

Zadanie:
W *n*-elementowej tablicy zawierającej litery znaleźć pierwsze wystąpienie litery „a”.

Rozwiązanie:
1. Losowe wybieranie miejsca w tablicy z zakresu od *0* do *n-1*,
2. Sprawdzenie, czy wybrane miejsce wskazuje literę „a”, jeśli nie to idź do 1,
   jeśli tak to idź do 3.
3. Wypisz pozycję „a” i zakończ wyszukiwanie.

Zadanie:
W *n*-elementowej tablicy zawierającej litery znaleźć pierwsze wystąpienie litery „a”.

Rozwiązanie:
1. Losowe wybieranie miejsca w tablicy z zakresu od *0* do *n-1*,
2. Sprawdzenie, czy wybrane miejsce wskazuje literę „a”, jeśli nie to idź do 1, jeśli tak to idź do 3.
3. Wypisz pozycję „a” i zakończ wyszukiwanie.

Załóżmy, że poszukiwana litera jest w tablicy. Wtedy, statystycznie potrzeba *n/2* prób, aby poszukiwaną literę znaleźć. Zatem losowe wybieranie elementów pozwoli po co najwyżej *n/2* próbach trafić w szukane miejsce.

Zadanie:
W *n*-elementowej tablicy zawierającej litery znaleźć pierwsze wystąpienie litery „a”.

Rozwiązanie:
1. Losowe wybieranie miejsca w tablicy z zakresu od *0* do *n-1*,
2. Sprawdzenie, czy wybrane miejsce wskazuje literę „a”, jeśli nie to idź do 1,
   jeśli tak to idź do 3.
3. Wypisz pozycję „a” i zakończ wyszukiwanie.

Załóżmy, że poszukiwana litera jest w tablicy. Wtedy, statystycznie potrzeba *n/2* prób, aby poszukiwaną literę znaleźć. Zatem losowe wybieranie elementów pozwoli po co najwyżej *n/2* próbach trafić w szukane miejsce.

Średni czas działania algorytmu deterministycznego dla „danych złośliwych” – pozycja litery „a” jest w pobliżu środka tablicy – wyniesie *n/2* prób.

# Koniec wykładu