Algorytmy i struktury danych

wykład 4

Plan wykładu:

- Klasy algorytmów – część druga.
- Techniki implementacji algorytmów.
- Złożoność obliczeniowa algorytmów:
  - praktyczna,
  - teoretyczna.
- Klasy złożoności obliczeniowej.
- Typy złożoności obliczeniowej.
- Problemy klasy P i NP.

Algorytmy heurystyczne

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Informacje podstawowe:

- wykorzystywane wszędzie tam, gdzie algorytmy dokładne są zbyt kosztowne,
- stosowane, gdy algorytm dokładny nie jest znany,
- są często stosowane do wyznaczania rozwiązań przybliżonych, które następnie optymalizuje się algorytmem dokładnym,
- heurystyki pozwalają często znaleźć dobre rozwiązanie, jeśli dopuszcza się pewien zakres rozwiązań akceptowalnych.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Informacje podstawowe:

- wykorzystywane wszędzie tam, gdzie algorytmy dokładne są zbyt kosztowne,
- stosowane, gdy algorytm dokładny nie jest znany,
- są często stosowane do wyznaczania rozwiązań przybliżonych, które następnie optymalizuje się algorytmem dokładnym,
- heurystyki pozwalają często znaleźć dobre rozwiązanie, jeśli dopuszcza się pewien zakres rozwiązań akceptowalnych.

→ Rozwiązywanie problemów NP-trudnych i NP-zupełnych.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Informacje podstawowe:

- wykorzystywane wszędzie tam, gdzie algorytmy dokładne są zbyt kosztowne,
- stosowane, gdy algorytm dokładny nie jest znany,
- są często stosowane do wyznaczania rozwiązań przybliżonych, które następnie optymalizuje się algorytmem dokładnym,
- heurystyki pozwalają często znaleźć dobre rozwiązanie, jeśli dopuszcza się pewien zakres rozwiązań akceptowalnych.

→ Rozwiązywanie problemów NP-trudnych i NP-zupełnych.

→ Algorytmy bardzo złożone lub scharakteryzowane ogromną liczbą zmiennych.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Informacje podstawowe:

- wykorzystywane wszędzie tam, gdzie algorytmy dokładne są zbyt kosztowne,
- stosowane, gdy algorytm dokładny nie jest znany,
- są często stosowane do wyznaczania rozwiązań przybliżonych, które następnie optymalizuje się algorytmem dokładnym,
- heurystyki pozwalają często znaleźć dobre rozwiązanie, jeśli dopuszcza się pewien zakres rozwiązań akceptowalnych.

→ Rozwiązywanie problemów NP-trudnych i NP-zupełnych.

→ Algorytmy bardzo złożone lub scharakteryzowane ogromną liczbą zmiennych.

→ Heurystyka jest wykorzystywana do nakierowywania pełnego algorytmu ku optymalnemu rozwiązaniu, co pozwala na zmniejszenie czas działania programu w typowym przypadku bez utraty jakości rozwiązania.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Informacje podstawowe:

- wykorzystywane wszędzie tam, gdzie algorytmy dokładne są zbyt kosztowne,
- stosowane, gdy algorytm dokładny nie jest znany,
- są często stosowane do wyznaczania rozwiązań przybliżonych, które następnie optymalizuje się algorytmem dokładnym,
- heurystyki pozwalają często znaleźć dobre rozwiązanie, jeśli dopuszcza się pewien zakres rozwiązań akceptowalnych.

→ Rozwiązywanie problemów NP-trudnych i NP-zupełnych.

→ Algorytmy bardzo złożone lub scharakteryzowane ogromną liczbą zmiennych.

→ Heurystyka jest wykorzystywana do nakierowywania pełnego algorytmu ku optymalnemu rozwiązaniu, co pozwala na zmniejszenie czas działania programu w typowym przypadku bez utraty jakości rozwiązania.

Metody algorytmiczne stosowane są najczęściej w przypadku zbadanych, znanych już problemów. Metody heurystyczne stosuje się natomiast wszędzie tam, gdzie typowe algorytmy nie wystarczają do rozwiązania zadania i poszukiwana jest nowa metoda lub sposób odnajdywania rozwiązania.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Różnice między postępowaniem algorytmicznym i heurystycznym:

- metody algorytmiczne :
  rozwiązanie zostanie zawsze znalezione, choć czas jego wyznaczenia może być nieskończenie długi,
- metody heurystyczne :
  czas znalezienia rozwiązania nie jest nieskończenie długi, jednak znalezione rozwiązanie może nie należeć do rozwiązań akceptowalnych.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Różnice między postępowaniem algorytmicznym i heurystycznym:

- metody algorytmiczne :
  rozwiązanie zostanie zawsze znalezione, choć czas jego wyznaczenia może być nieskończenie długi,
- metody heurystyczne :
  czas znalezienia rozwiązania nie jest nieskończenie długi, jednak znalezione rozwiązanie może nie należeć do rozwiązań akceptowalnych.

Założenia heurystyk:

- zasada kompletności,
- zasada relewantności.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Różnice między postępowaniem algorytmicznym i heurystycznym:

- metody algorytmiczne :
  rozwiązanie zostanie zawsze znalezione, choć czas jego wyznaczenia może być nieskończenie długi,
- metody heurystyczne :
  czas znalezienia rozwiązania nie jest nieskończenie długi, jednak znalezione rozwiązanie może nie należeć do rozwiązań akceptowalnych.

Założenia heurystyk:

- zasada kompletności,
- zasada relewantności.

Definiowany przez użytkownika poziom trafności i kompletności informacji, jaki jest poszukiwany – dokładne określenie do czego informacja jest potrzebna.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Różnice między postępowaniem algorytmicznym i heurystycznym:

- metody algorytmiczne :
  rozwiązanie zostanie zawsze znalezione, choć czas jego wyznaczenia może być nieskończenie długi,
- metody heurystyczne :
  czas znalezienia rozwiązania nie jest nieskończenie długi, jednak znalezione rozwiązanie może nie należeć do rozwiązań akceptowalnych.

Założenia heurystyk:

- zasada kompletności,
- zasada relewantności.

Parametr jest powiązany z zasadą kompletności informacji. Zwiększanie ważność danego rozwiązania powoduje zazwyczaj obniżanie możliwości uzyskania rozwiązania kompletnego.

Definiowany przez użytkownika poziom trafności i kompletności informacji, jaki jest poszukiwany – dokładne określenie do czego informacja jest potrzebna.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Strategia wyszukiwania:

- wybraną metodę rozwiązania problemu traktuje się jako hipotezę,
- początkowe wyniki traktuje się jako prawdopodobnie dobre rozwiązania, ale poszukuje się innych o zbliżonych właściwościach,
- należy zastosować wszelkie alternatywne metody wyszukiwania rozwiązania,
- zbiór potencjalnych rozwiązań może mieć przypadkowe rozłożenie.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Strategia wyszukiwania:

- wybraną metodę rozwiązania problemu traktuje się jako hipotezę,
- początkowe wyniki traktuje się jako prawdopodobnie dobre rozwiązania, ale poszukuje się innych o zbliżonych właściwościach,
- należy zastosować wszelkie alternatywne metody wyszukiwania rozwiązania,
- zbiór potencjalnych rozwiązań może mieć przypadkowe rozłożenie.

→ Nie ma pewności, że dana metoda heurystyczna jest optymalna.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Strategia wyszukiwania:

- wybraną metodę rozwiązania problemu traktuje się jako hipotezę,
- początkowe wyniki traktuje się jako prawdopodobnie dobre rozwiązania, ale poszukuje się innych o zbliżonych właściwościach,
- należy zastosować wszelkie alternatywne metody wyszukiwania rozwiązania,
- zbiór potencjalnych rozwiązań może mieć przypadkowe rozłożenie.

→ Nie ma pewności, że dana metoda heurystyczna jest optymalna.

→ W heurystykach można znaleźć rozwiązania akceptowalne.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Strategia wyszukiwania:

- wybraną metodę rozwiązania problemu traktuje się jako hipotezę, → Nie ma pewności, że dana metoda heurystyczna jest optymalna.
- początkowe wyniki traktuje się jako prawdopodobnie dobre rozwiązania, ale poszukuje się innych o zbliżonych właściwościach, → W heurystykach można znaleźć rozwiązania akceptowalne.
- należy zastosować wszelkie alternatywne metody wyszukiwania rozwiązania, → Każde rozwiązanie heurystyczne jest przybliżone, dlatego inna metoda wyszukiwania może zaowocować pozyskaniem znacznie lepszych rozwiązań, niż stosowana do tej pory strategia wyszukiwania.
- zbiór potencjalnych rozwiązań może mieć przypadkowe rozłożenie.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Strategia wyszukiwania:

- wybraną metodę rozwiązania problemu traktuje się jako hipotezę,
- początkowe wyniki traktuje się jako prawdopodobnie dobre rozwiązania, ale poszukuje się innych o zbliżonych właściwościach,
- należy zastosować wszelkie alternatywne metody wyszukiwania rozwiązania,
- zbiór potencjalnych rozwiązań może mieć przypadkowe rozłożenie.

→ Nie ma pewności, że dana metoda heurystyczna jest optymalna.

→ W heurystykach można znaleźć rozwiązania akceptowalne.

→ Każde rozwiązanie heurystyczne jest przybliżone, dlatego inna metoda wyszukiwania może zaowocować pozyskaniem znacznie lepszych rozwiązań, niż stosowana do tej pory strategia wyszukiwania.

→ Nie należy zakładać, że początkowe uporządkowanie zbioru rozwiązań jest optymalne – wyniki lub rozwiązania nie powinny zależeć od kolejności elementów przeszukiwanego zbioru.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Skuteczność wyszukiwania:

kompletność rozwiązania	dokładność wyszukiwania	wskaźnik rozwiązań błędnych	trafność rozwiązań

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Skuteczność wyszukiwania:

kompletność rozwiązania

dokładność wyszukiwania

wskaźnik rozwiązań błędnych

trafność rozwiązań

Wskaźnik liczby pozyskanych akceptowalnych rozwiązań w stosunku do liczby wszystkich rozwiązań.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Skuteczność wyszukiwania:

kompletność rozwiązania

dokładność wyszukiwania

wskaźnik rozwiązań błędnych

trafność rozwiązań

Wskaźnik liczby pozyskanych akceptowalnych rozwiązań w stosunku do liczby wszystkich rozwiązań.

Wskaźnik dopasowania rozwiązania do oczekiwanej odpowiedzi – obliczane jako stosunek liczby wyników akceptowalnych do wszystkich zwracanych rozwiązań.

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Skuteczność wyszukiwania:

kompletność rozwiązania	dokładność wyszukiwania	wskaźnik rozwiązań błędnych	trafność rozwiązań
Wskaźnik liczby pozyskanych akceptowalnych rozwiązań w stosunku do liczby wszystkich rozwiązań.	Wskaźnik dopasowania rozwiązania do oczekiwanej odpowiedzi – obliczane jako stosunek liczby wyników akceptowalnych do wszystkich zwracanych rozwiązań.	Wskaźnik określa liczbę wyników nie spełniających wymagań.	

Algorytmy heurystyczne – jest to klasa algorytmów, które ograniczają przestrzeń poszukiwań do zbiorów, w których znalezienie optymalnych i akceptowalnych rozwiązań jest najbardziej prawdopodobne.

Skuteczność wyszukiwania:

kompletność rozwiązania	dokładność wyszukiwania	wskaźnik rozwiązań błędnych	trafność rozwiązań
Wskaźnik liczby pozyskanych akceptowalnych rozwiązań w stosunku do liczby wszystkich rozwiązań.	Wskaźnik dopasowania rozwiązania do oczekiwanej odpowiedzi – obliczane jako stosunek liczby wyników akceptowalnych do wszystkich zwracanych rozwiązań.	Wskaźnik określa liczbę wyników nie spełniających wymagań.	Wskaźnik określa stopień trafności wyniku. Wskaźnik jest subiektywnie definiowany przez użytkownika.

Zadanie:
Znaleźć najkrótszą trasę między miastami, przechodząc przez wszystkie miasta.

Cechy zadania:
- rozpatrywany problem jest NP-trudny,
- czas działania algorytmu siłowego dla więcej niż 15 miast nie jest akceptowalny.

Zadanie:
Znaleźć najkrótszą trasę między miastami, przechodząc przez wszystkie miasta.

Cechy zadania:
- rozpatrywany problem jest NP-trudny,
- czas działania algorytmu siłowego dla więcej niż 15 miast nie jest akceptowalny.

Heurystyczne uproszczenia problemu:
- miasta i drogi tworzą graf planarny (są położone na płaszczyźnie),
- w danym obszarze rozłożenie miast jest w miarę równomierne,
- miasta mają tendencję do klastrowania się, czyli algorytm można wpierw wykonać dla klastrów, a potem ponownie wykonać go dla każdego klastra z osobna,
- można szybko oszacować odległości między miastami, tak jakby łączyły je linie proste,
- trasy nie powinny krzyżować się,
- trasa powinna zaczynać się na brzegu obszaru, a poruszanie się powinno okrążać obszar – nie należy poruszać się chaotycznie,
- można wybrać trasę losowo i następnie starać się heurystycznie polepszyć jej jakość.

Zadanie:
Znaleźć najkrótszą trasę między miastami, przechodząc przez wszystkie miasta.

Cechy zadania:
- rozpatrywany problem jest NP-trudny,
- czas działania algorytmu siłowego dla więcej niż 15 miast nie jest akceptowalny.

Heurystyczne uproszczenia problemu:
- miasta i drogi tworzą graf planarny (są położone na płaszczyźnie),
- w danym obszarze rozłożenie miast jest w miarę równomierne,
- miasta mają tendencję do klastrowania się, czyli algorytm można wpierw wykonać dla klastrów, a potem ponownie wykonać go dla każdego klastra z osobna,
- można szybko oszacować odległości między miastami, tak jakby łączyły je linie proste,
- trasy nie powinny krzyżować się,
- trasa powinna zaczynać się na brzegu obszaru, a poruszanie się powinno okrążać obszar – nie należy poruszać się chaotycznie,
- można wybrać trasę losowo i następnie starać się heurystycznie polepszyć jej jakość.

Zastosowanie heurystyk pozwala:
- zwiększyć liczbę miast,
- znacznie przyspieszyć algorytm, nawet dla dużej liczby miast.

Symulowane wyżarzanie

Symulowane wyżarzanie – jest to rodzaj algorytmu heurystycznego przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Konstrukcja algorytmu polega na znajdowaniu wyznaczeniu funkcji „dobroci” rozwiązania, a następnie na jej ulepszaniu – stąd w nazwie „wyżarzanie”.

Symulowane wyżarzanie – jest to rodzaj algorytmu heurystycznego przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Konstrukcja algorytmu polega na znajdowaniu wyznaczeniu funkcji „dobroci” rozwiązania, a następnie na jej ulepszaniu – stąd w nazwie „wyżarzanie”.

Etapy minimalizacji funkcji F(ω):

1. Losowy wybór punktu startowego ω. Przyjęcie temperatury $T = T_{max}$.
2. Wyznaczenie wartości funkcji F w punkcje ω.
3. Wyznaczenie $\omega' = \omega + \Delta\omega$, gdzie $\Delta\omega$ jest realizacją zmiennej losowej o rozkładzie normalnym z medianą w punkcje $\Delta\omega$ i średnią wariancją równą T.
4. Wyznaczenie wartości funkcji F w punkcje ω'.
5. Podstawienie wartości ω' do ω z prawdopodobieństwem danym rozkładem Boltzmanna $b(E(\omega') - E(\omega), T)$.
6. Zmniejszenie temperatury $T = nT$, gdzie n jest stałą z przedziału (0,1).
7. Ponowne wykonanie algorytmu od kroku 3.

Symulowane wyżarzanie – jest to rodzaj algorytmu heurystycznego przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Konstrukcja algorytmu polega na znajdowaniu wyznaczeniu funkcji „dobroci” rozwiązania, a następnie na jej ulepszaniu – stąd w nazwie „wyżarzanie”.

Etapy minimalizacji funkcji F(ω):

1. Losowy wybór punktu startowego ω. Przyjęcie temperatury $T = T_{max}$.
2. Wyznaczenie wartości funkcji F w punkcje ω.
3. Wyznaczenie $\omega' = \omega + \Delta\omega$, gdzie $\Delta\omega$ jest realizacją zmiennej losowej o rozkładzie normalnym z medianą w punkcje $\Delta\omega$ i średnią wariancją równą T.
4. Wyznaczenie wartości funkcji F w punkcje ω'.
5. Podstawienie wartości ω' do ω z prawdopodobieństwem danym rozkładem Boltzmanna $b(E(\omega') - E(\omega), T)$.
6. Zmniejszenie temperatury $T = nT$, gdzie n jest stałą z przedziału (0,1).
7. Ponowne wykonanie algorytmu od kroku 3.

Wyżarzanie funkcji F kończy się, gdy znalezione w ten sposób rozwiązanie problemu spełnia oczekiwania.

Zadanie:
Znaleźć najkrótszą trasę między miastami, przechodząc przez wszystkie miasta.

Założenia:
- miasta oznaczamy jako $m_1, m_2, \ldots m_n$,
- trasa będzie ciągiem miast (permutacja zbioru miast), np.: $m_1 - m_2 - m_5 - m_7 - \ldots - m_n$.

Zadanie:
Znaleźć najkrótszą trasę między miastami, przechodząc przez wszystkie miasta.

Założenia:
- miasta oznaczamy jako $m_1, m_2, \ldots m_n$,
- trasa będzie ciągiem miast (permutacja zbioru miast), np.: $m_1 - m_2 - m_5 - m_7 - \ldots - m_n$.

Rozwiązanie:
1. Wybranie dowolnej permutacji *n* miast.
2. Wykonanie próbnej permutacji dwóch miast w zbiorze wybranym na etapie 1:
   - Jeśli zmiana spowodowała skrócenie drogi, to jest akceptowana.
   - Jeśli zmiana nie powoduje skrócenie drogi, to może być akceptowana z uwzględnieniem prawdopodobieństwa, zależnym od wielkości wydłużenia – im większe wydłużenie tym mniejsze prawdopodobieństwo.
3. Ponowne wykonanie etapu 2, zmniejszając stopniowo możliwość wydłużenia (schładzanie).

Algorytmy genetyczne

Algorytmy genetyczne – jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Informacje podstawowe:

- problem definiuje środowisko,
- definiuje się populację osobników,
- osobnik posiada genotyp,
- środowisko posiada fenotyp.

Algorytmy genetyczne – jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Informacje podstawowe:

- problem definiuje środowisko,
- definiuje się populację osobników,
- osobnik posiada genotyp,
- środowisko posiada fenotyp.

→ W środowisku (przestrzeni rozwiązań) istnieje populacja osobników – rozwiązań.

Algorytmy genetyczne – jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Informacje podstawowe:

- problem definiuje środowisko,
- definiuje się populację osobników,
- osobnik posiada genotyp,
- środowisko posiada fenotyp.

W środowisku (przestrzeni rozwiązań) istnieje populacja osobników – rozwiązań.

Zbiór informacji charakteryzujących dane rozwiązanie – osobnika. Genotyp składa się z chromosomów, gdzie zakodowany jest fenotyp i ewentualnie pewne dodatkowe informacje pomocnicze. Chromosom zawiera geny.

Algorytmy genetyczne – jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Informacje podstawowe:

- problem definiuje środowisko,
- definiuje się populację osobników,
- osobnik posiada genotyp,
- środowisko posiada fenotyp.

W środowisku (przestrzeni rozwiązań) istnieje populacja osobników – rozwiązań.

Zbiór informacji charakteryzujących dane rozwiązanie – osobnika. Genotyp składa się z chromosomów, gdzie zakodowany jest fenotyp i ewentualnie pewne dodatkowe informacje pomocnicze. Chromosom zawiera geny.

Zbiór cech podlegających ocenie funkcji przystosowania modelującej środowisko.

Algorytmy genetyczne – jest to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Informacje podstawowe:

- problem definiuje środowisko,
- definiuje się populację osobników,
- osobnik posiada genotyp,
- środowisko posiada fenotyp.

W środowisku (przestrzeni rozwiązań) istnieje populacja osobników – rozwiązań.

Zbiór informacji charakteryzujących dane rozwiązanie – osobnika. Genotyp składa się z chromosomów, gdzie zakodowany jest fenotyp i ewentualnie pewne dodatkowe informacje pomocnicze. Chromosom zawiera geny.

Zbiór cech podlegających ocenie funkcji przystosowania modelującej środowisko.

Genotyp opisuje proponowane rozwiązanie problemu, a funkcja przystosowania ocenia, jak dobre jest to rozwiązanie.

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Realizuje proces wyboru osobników poddanych ocenie według określonych kryteriów. Zwykle algorytm genetyczny dąży do minimalizacji lub maksymalizacji funkcji przystosowania.

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Realizuje proces wyboru osobników poddanych ocenie według określonych kryteriów. Zwykle algorytm genetyczny dąży do minimalizacji lub maksymalizacji funkcji przystosowania.

Jest to miara jakości dowolnego osobnika (fenotypu, rozwiązania) w populacji. Dla każdego osobnika jest ona obliczana na podstawie pewnego modelu rozwiązywanego problemu

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Realizuje proces wyboru osobników poddanych ocenie według określonych kryteriów. Zwykle algorytm genetyczny dąży do minimalizacji lub maksymalizacji funkcji przystosowania.

Jest to miara jakości dowolnego osobnika (fenotypu, rozwiązania) w populacji. Dla każdego osobnika jest ona obliczana na podstawie pewnego modelu rozwiązywanego problemu

Cel:
Wygenerowanie nowego pokolenia, na podstawie poprzedniego, które **być może** będzie lepiej dopasowane do założonego środowiska.

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Realizuje proces wyboru osobników poddanych ocenie według określonych kryteriów. Zwykle algorytm genetyczny dąży do minimalizacji lub maksymalizacji funkcji przystosowania.

Jest to miara jakości dowolnego osobnika (fenotypu, rozwiązania) w populacji. Dla każdego osobnika jest ona obliczana na podstawie pewnego modelu rozwiązywanego problemu

Cel:
Wygenerowanie nowego pokolenia, na podstawie poprzedniego, które **być może** będzie lepiej dopasowane do założonego środowiska.

Krzyżowanie:
Wyznaczanie różnych kombinacji cech pochodzących od różnych osobników populacji. Krzyżowanie jest sposobem eksploatację przestrzeni rozwiązań.

Pojęcia związane z algorytmami genetycznymi:

- funkcja przystosowania,
- funkcja oceny,
- operatory przeszukiwania:
  - operator krzyżowania,
  - operator mutacji.

Realizuje proces wyboru osobników poddanych ocenie według określonych kryteriów. Zwykle algorytm genetyczny dąży do minimalizacji lub maksymalizacji funkcji przystosowania.

Jest to miara jakości dowolnego osobnika (fenotypu, rozwiązania) w populacji. Dla każdego osobnika jest ona obliczana na podstawie pewnego modelu rozwiązywanego problemu

Cel:
Wygenerowanie nowego pokolenia, na podstawie poprzedniego, które **być może** będzie lepiej dopasowane do założonego środowiska.

Krzyżowanie:
Wyznaczanie różnych kombinacji cech pochodzących od różnych osobników populacji. Krzyżowanie jest sposobem eksploatację przestrzeni rozwiązań.

Mutacja:
Zwiększanie różnorodności osobników – jest to sposób na eksplorację przestrzeni rozwiązań.

Wspólne cechy algorytmów ewolucyjnych:

1. stosowanie operatorów genetycznych, które dostosowane są do postaci rozwiązań,
2. przetwarzanie populacji rozwiązań, prowadzące do równoległego przeszukiwania przestrzeni rozwiązań z różnych punktów,
3. w celu ukierunkowania procesu przeszukiwania wystarczającą informacją jest jakość aktualnych rozwiązań,
4. celowe wprowadzenie elementów losowych.

Wspólne cechy algorytmów ewolucyjnych:

1. stosowanie operatorów genetycznych, które dostosowane są do postaci rozwiązań,
2. przetwarzanie populacji rozwiązań, prowadzące do równoległego przeszukiwania przestrzeni rozwiązań z różnych punktów,
3. w celu ukierunkowania procesu przeszukiwania wystarczającą informacją jest jakość aktualnych rozwiązań,
4. celowe wprowadzenie elementów losowych.

Ogólny schemat algorytmu:

1. Losowanie populacji początkowej.
2. Selekcja najlepiej rokujących osobników wyznaczonych na etapie 1.
3. Zastosowanie do genotypów wyselekcjonowanych osobników operatorów ewolucyjnych:
   a. łączenie genotypów „rodziców” - krzyżowanie.
   b. wprowadzenie mutacji – czyli implementacja drobnych, losowych zmian.
4. Wyznaczenie nowego pokolenia:
   a. usunięcie osobników najsłabszych (do oceny stosowana jest funkcja oceny fenotypu),
   b. jeśli nie znaleziono osobnika o akceptowalnych cechach – idź do 2,
      jeśli odpowiedni osobnik został znaleziony, to jego genotyp jest rozwiązaniem problemu.

Algorytmy kwantowe

Algorytmy kwantowe – algorytm przeznaczony do działania na maszynie kwantowej, przystosowany do specyfiki działania komputera kwantowego.

Maszyna kwantowa:

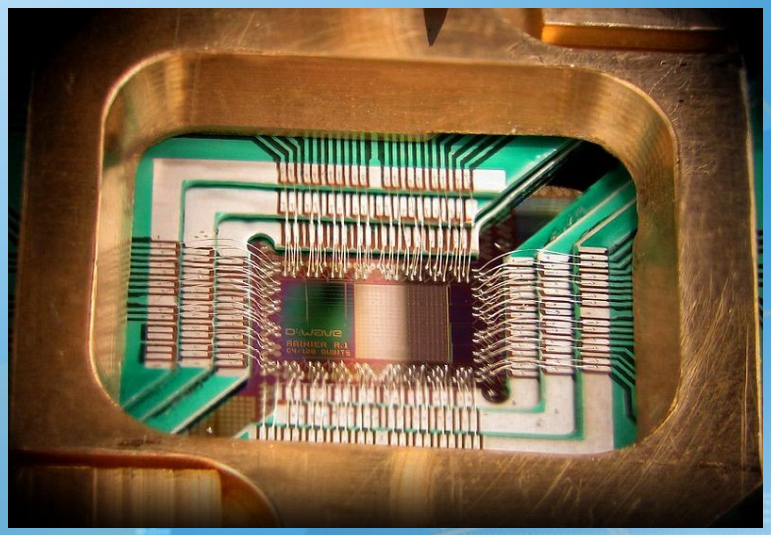
- qubit – jednostka informacji w informatyce kwantowej,
- superpozycja stanów,
- uwikłanie.

Algorytmy kwantowe – algorytm przeznaczony do działania na maszynie kwantowej, przystosowany do specyfiki działania komputera kwantowego.

Maszyna kwantowa:

- qubit – jednostka informacji w informatyce kwantowej,
- superpozycja stanów,
- uwikłanie.

Odpowiednik bitu w systemach informatyki klasycznej. Dla qubitów zdefiniowane qubitowe bramki logiczne – odpowiedniki klasycznych bramek logicznych.



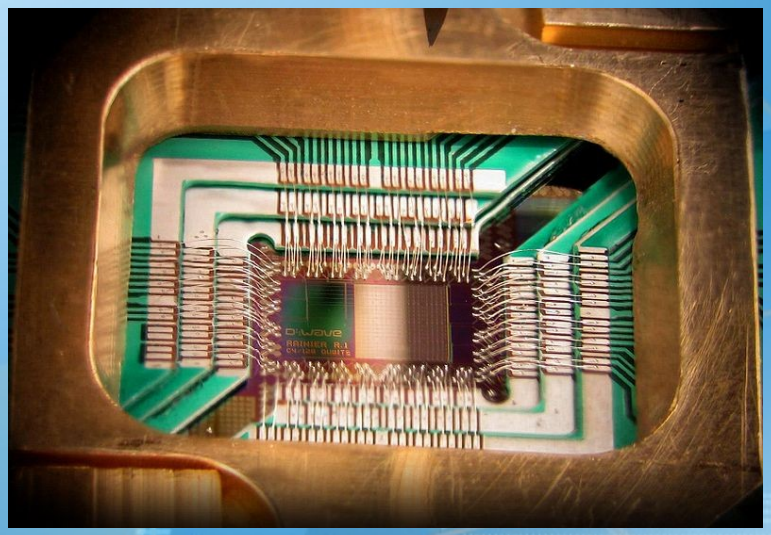
Układ D-wave Systems (128 qubitów)

Algorytmy kwantowe – algorytm przeznaczony do działania na maszynie kwantowej, przystosowany do specyfiki działania komputera kwantowego.

Maszyna kwantowa:

- qubit – jednostka informacji w informatyce kwantowej,
- superpozycja stanów,
- uwikłanie.

Odpowiednik bitu w systemach informatyki klasycznej. Dla qubitów zdefiniowane qubitowe bramki logiczne – odpowiedniki klasycznych bramek logicznych.



Układ D-wave Systems (128 qubitów)

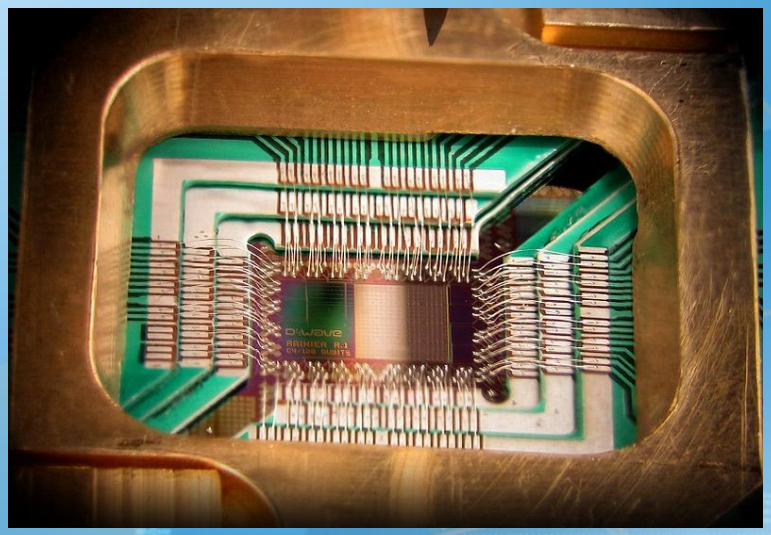
Zjawisko uwikłania (splątania) związane jest z dowolnym układem n-qubitów. Powoduje, że układy kwantowe wchodzą ze sobą we wzajemne związki, które w komputerze kwantowym są odpowiednikiem przewodów łączących poszczególne qubity,

Algorytmy kwantowe – algorytm przeznaczony do działania na maszynie kwantowej, przystosowany do specyfiki działania komputera kwantowego.

Maszyna kwantowa:

- qubit – jednostka informacji w informatyce kwantowej,
- superpozycja stanów,
- uwikłanie.

Odpowiednik bitu w systemach informatyki klasycznej. Dla qubitów zdefiniowane qubitowe bramki logiczne – odpowiedniki klasycznych bramek logicznych.



Układ D-wave Systems (128 qubitów)

N-qubitowy komputer kwantowy może istnieć w wielu stanach jednocześnie, dzięki czemu można równolegle wykonywać operacje kwantowe na każdym ze stanów.

Zjawisko uwikłania (splątania) związane jest z dowolnym układem n-qubitów. Powoduje, że układy kwantowe wchodzą ze sobą we wzajemne związki, które w komputerze kwantowym są odpowiednikiem przewodów łączących poszczególne qubity,

Specyfika algorytmów kwantowych:

- uwzględniają specyfikę systemu kwantowego,
- stosowane są głównie w problemach obliczeniowych,
- zrównoleglenie rozwiązań pewnej klasy problemów:
  - pozwalają na znaczne przyspieszenie algorytmów informatyki klasycznej,
  - otwierają drogę do kryptografii kwantowej.

Specyfika algorytmów kwantowych:

- uwzględniają specyfikę systemu kwantowego,
- stosowane są głównie w problemach obliczeniowych,
- zrównoleglenie rozwiązań pewnej klasy problemów:
  - pozwalają na znaczne przyspieszenie algorytmów informatyki klasycznej,
  - otwierają drogę do kryptografii kwantowej.

Przykłady algorytmów:

- algorytm Shora – służy do znajdowania liczb pierwszych,
- algorytm Kitajewa – szybka kwantowa transformacja Fouriera,
- algorytm Grovera – przeszukiwania bazy danych, klasyczne algorytmy potrzebują średnio $n/2$ kroków, algorytm kwantowy wymaga $\sqrt{n}$ kroków.

Specyfika algorytmów kwantowych:

- uwzględniają specyfikę systemu kwantowego,
- stosowane są głównie w problemach obliczeniowych,
- zrównoleglenie rozwiązań pewnej klasy problemów:
  - pozwalają na znaczne przyspieszenie algorytmów informatyki klasycznej,
  - otwierają drogę do kryptografii kwantowej.

Przykłady algorytmów:

- algorytm Shora – służy do znajdowania liczb pierwszych,
- algorytm Kitajewa – szybka kwantowa transformacja Fouriera,
- algorytm Grovera – przeszukiwania bazy danych, klasyczne algorytmy potrzebują średnio $n/2$ kroków, algorytm kwantowy wymaga $\sqrt{n}$ kroków.

Realizacje praktyczne:

2001 – w IBM dokonano rozkładu liczby 15 na 3·5 na 7-qubitowej maszynie kwantowej,
2007 – zaprezentowano 128-qubitowy rejestr kwantowy,
2011 – dokonano faktoryzacji liczby 21,
2012 – na 84-qubitowym komputerem kwantowym obliczono kilka liczb Ramseya.

Techniki implementacji algorytmów

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Polega na podziale implementowanego algorytmu na podprogramu

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Polega na podziale implementowanego algorytmu na podprogramu

Polega na szeregowym wykonaniu procedur algorytmu – w danej chwili pracuje tylko jedna procedura, a procesor wykonuje tylko jeden etap algorytmu.

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Polega na podziale implementowanego algorytmu na podprogramu

Polega na szeregowym wykonaniu procedur algorytmu – w danej chwili pracuje tylko jedna procedura, a procesor wykonuje tylko jeden etap algorytmu.

Kolejność wykonania procedur algorytmu jest zgodna ze strategią szeregowania zadań systemu operacyjnego, wymaga synchronizacji między procedurami wykonującymi kolejne etapy algorytmu.

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Polega na podziale implementowanego algorytmu na podprogramu

Polega na szeregowym wykonaniu procedur algorytmu – w danej chwili pracuje tylko jedna procedura, a procesor wykonuje tylko jeden etap algorytmu.

Umożliwia jednoczesne wykonanie wielu etapów algorytmów, wymaga specjalnego zaprojektowania algorytmu i odpowiedniej synchronizacji.

Kolejność wykonania procedur algorytmu jest zgodna ze strategią szeregowania zadań systemu operacyjnego, wymaga synchronizacji między procedurami wykonującymi kolejne etapy algorytmu.

Sposoby implementacji algorytmów:

- wykorzystanie procedur,
- wykonanie sekwencyjne,
- praca wielowątkowa,
- praca równoległa,
- rekurencje.

Polega na podziale implementowanego algorytmu na podprogramu

Polega na szeregowym wykonaniu procedur algorytmu – w danej chwili pracuje tylko jedna procedura, a procesor wykonuje tylko jeden etap algorytmu.

Kolejność wykonania procedur algorytmu jest zgodna ze strategią szeregowania zadań systemu operacyjnego, wymaga synchronizacji między procedurami wykonującymi kolejne etapy algorytmu.

Umożliwia jednoczesne wykonanie wielu etapów algorytmów, wymaga specjalnego zaprojektowania algorytmu i odpowiedniej synchronizacji.

Wykonanie algorytmu w postaci procedur wywołujących same siebie z odpowiednimi parametrami.

Złożoność obliczeniowa algorytmów

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa, → Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa, → Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.
- złożoność pamięciowa, → Liczba bajtów, kilobajtów, megabajtów pamięci RAM i/lub pamięci dyskowej. Również liczba plików, łącz nazwanych/nienazwanych i innych zasobów pamięciowych jakie są niezbędne do realizacji algorytmu.
- specyfikacja komputera,
- charakter algorytmu.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.

Liczba bajtów, kilobajtów, megabajtów pamięci RAM i/lub pamięci dyskowej. Również liczba plików, łącz nazwanych/nienazwanych i innych zasobów pamięciowych jakie są niezbędne do realizacji algorytmu.

Rodzaj i typ mikroprocesora. Liczba rdzenie, specyficzne właściwości, takie jak możliwość pracy równoległej. Typ systemu komputerowego: SISD, SIMD itp., ilość dostępnej pamięci, szybkość pamięci, szybkość pamięci masowej.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.

Liczba bajtów, kilobajtów, megabajtów pamięci RAM i/lub pamięci dyskowej. Również liczba plików, łącz nazwanych/nienazwanych i innych zasobów pamięciowych jakie są niezbędne do realizacji algorytmu.

Algorytm można zaprojektować jako:
- szeregową sieć działań,
- rozwiązanie dla pracy równoległej,
- o mniejszych lub większych wymaganiach pamięciowych.

Rodzaj i typ mikroprocesora. Liczba rdzenie, specyficzne właściwości, takie jak możliwość pracy równoległej. Typ systemu komputerowego: SISD, SIMD itp., ilość dostępnej pamięci, szybkość pamięci, szybkość pamięci masowej.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Liczba sekund, godzin, dni .. najczęściej liczba operacji jakie należy wykonać, aby problem został rozwiązany.

Liczba bajtów, kilobajtów, megabajtów pamięci RAM i/lub pamięci dyskowej. Również liczba plików, łącz nazwanych/nienazwanych i innych zasobów pamięciowych jakie są niezbędne do realizacji algorytmu.

Algorytm można zaprojektować jako:
- szeregową sieć działań,
- rozwiązanie dla pracy równoległej,
- o mniejszych lub większych wymaganiach pamięciowych.

Rodzaj i typ mikroprocesora. Liczba rdzenie, specyficzne właściwości, takie jak możliwość pracy równoległej. Typ systemu komputerowego: SISD, SIMD itp., ilość dostępnej pamięci, szybkość pamięci, szybkość pamięci masowej.

Złożoność obliczeniowa jest uzależniona od typu algorytmu oraz od architektury systemu komputerowego na którym dany problem jest rozwiązywany.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:

- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Powyższa specyfika powoduje, że nie określa się złożoności obliczeniowej podając czas.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:
- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Powyższa specyfika powoduje, że nie określa się złożoności obliczeniowej podając czas.

Miary związane ze złożonością obliczeniową:
- mierzalne w danym systemie jednostki,
- liczba parametrów wejściowych *n*.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:
- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Powyższa specyfika powoduje, że nie określa się złożoności obliczeniowej podając czas.

Miary związane ze złożonością obliczeniową:
- mierzalne w danym systemie jednostki, → Na przykład cykle mikroprocesora.
- liczba parametrów wejściowych *n*.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Określenie wydajności rozwiązania:
- złożoność czasowa,
- złożoność pamięciowa,
- specyfikacja komputera,
- charakter algorytmu.

Powyższa specyfika powoduje, że nie określa się złożoności obliczeniowej podając czas.

Miary związane ze złożonością obliczeniową:
- mierzalne w danym systemie jednostki, → Na przykład cykle mikroprocesora.
- liczba parametrów wejściowych *n*.

Na przykład parametrem wejściowym może być **tablica**, wtedy parametrem *n* będzie tu liczba jej elementów, może to być **wektor**, **rekord**.

Pojęcie wielkości *n* jest względne – tablica o 10000 elementów wydaje się tablicą o dużej liczbie elementów, natomiast obliczenia dla *n* = 30 nie wydają się złożone, ale jeśli dotyczy to silni, to algorytm jest bardzo złożony.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Algorytm	**Złożoność czasowa**	**Czas przetwarzania *n* danych**
A1	n	0.2 ms

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Algorytm	**Złożoność czasowa**	**Czas przetwarzania *n* danych**
A1	n	0.2 ms
A2	n log n	3.5 ms

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Algorytm	**Złożoność czasowa**	**Czas przetwarzania *n* danych**
A1	n	0.2 ms
A2	n log n	3.5 ms
A3	n^2	40 s

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Algorytm	**Złożoność czasowa**	**Czas przetwarzania *n* danych**
A1	n	0.2 ms
A2	n log n	3.5 ms
A3	n^2	40 s
A4	n^3	3 miesiące

Złożoność obliczeniowa algorytmu – jest to miara określająca zapotrzebowania danego rozwiązania na zasoby systemu w którym ma być rozwiązany.

Złożoność obliczeniowa jest **tylko** pośrednio powiązana z czasem. Załóżmy, że program o danej złożoności obliczeniowej wykonuje się *x* sekund.

Algorytm	**Złożoność czasowa**	**Czas przetwarzania *n* danych**
A1	n	0.2 ms
A2	n log n	3.5 ms
A3	n^2	40 s
A4	n^3	3 miesiące
A5	2^n	36 mln lat

Złożoność obliczeniowa praktyczna

Rozważmy funkcję obliczającą silnię:

Algorytm:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Rozważmy funkcję obliczającą silnię:

Algorytm:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Rozważmy funkcję obliczającą silnię:

Algorytm:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Problem został rozwiązany rekurencyjnie.

Rozważmy funkcję obliczającą silnię:

Algorytm:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Problem został rozwiązany rekurencyjnie.

Zadanie:

Jak oszacować czas potrzebny na wykonanie algorytmu ?

Rozważmy funkcję obliczającą silnię:

Algorytm:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Problem został rozwiązany rekurencyjnie.

Zadanie:

Jak oszacować czas potrzebny na wykonanie algorytmu ?

Odpowiedź:

Trzeba znaleźć złożoność obliczeniową algorytmu.

Wyznaczanie złożoności obliczeniowej:

1. Określenie operacji najbardziej czasochłonnej i/lub wykonywanej najczęściej.
2. Określenie operacji dominującej.

Wyznaczanie złożoności obliczeniowej:

1. Określenie operacji najbardziej czasochłonnej i/lub wykonywanej najczęściej.
2. Określenie operacji dominującej.

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Wyznaczanie złożoności obliczeniowej:

1. Określenie operacji najbardziej czasochłonnej i/lub wykonywanej najczęściej.
2. Określenie operacji dominującej.

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Operacje najbardziej czasochłonne: mnożenie i odejmowanie.

Wyznaczanie złożoności obliczeniowej:

1. Określenie operacji najbardziej czasochłonnej i/lub wykonywanej najczęściej.
2. Określenie operacji dominującej.

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Operacje najbardziej czasochłonne: mnożenie i odejmowanie.

Operacja dominująca: porównanie.

Wyznaczanie złożoności obliczeniowej:

1. Określenie operacji najbardziej czasochłonnej i/lub wykonywanej najczęściej.
2. Określenie operacji dominującej.

Pseudokod algorytmu:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Operacje najbardziej czasochłonne: mnożenie i odejmowanie.

Operacja dominująca: porównanie.

Dla czasu wykonania operacji dominującej t_c:

$T(0) = t_c$ dla n = 0

$T(n) = t_c + T(n – 1)$ dla n > 0

Rozwiązanie nierekurencyjne:

$$T(n) = t_c + T(n - 1)$$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:

t_c – czas wykonania operacji dominującej.

Rozwiązanie nierekurencyjne:

$T(n) = t_c + T(n - 1)$
$T(n - 1) = t_c + T(n - 2)$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:

t_c – czas wykonania operacji dominującej.

Rozwiązanie nierekurencyjne:

$T(n) = t_c + T(n - 1)$
$T(n - 1) = t_c + T(n - 2)$
$T(n - 2) = t_c + T(n - 3)$
...
$T(1) = t_c + T(0)$
$T(0) = t_c$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:

t_c – czas wykonania operacji dominującej.

Rozwiązanie nierekurencyjne:

$T(n) = t_c + T(n-1)$
$T(n-1) = t_c + T(n-2)$
$T(n-2) = t_c + T(n-3)$
...
$T(1) = t_c + T(0)$
$T(0) = t_c$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:
t_c – czas wykonania operacji dominującej.

Po przekształceniu:

$$T(n) + T(n-1) + \ldots + T(0) = (n+1)\,t_c + T(n-1) + \ldots + T(0)$$

Rozwiązanie nierekurencyjne:

$T(n) = t_c + T(n-1)$
$T(n-1) = t_c + T(n-2)$
$T(n-2) = t_c + T(n-3)$
...
$T(1) = t_c + T(0)$
$T(0) = t_c$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:
t_c – czas wykonania operacji dominującej.

Po przekształceniu:

$$T(n) + T(n-1) + \ldots + T(0) = (n+1)\,t_c + T(n-1) + \ldots + T(0)$$

Po uproszczeniu:

$$T(n) = (n+1)\,t_c$$

Rozwiązanie nierekurencyjne:

$T(n) = t_c + T(n-1)$
$T(n-1) = t_c + T(n-2)$
$T(n-2) = t_c + T(n-3)$
...
$T(1) = t_c + T(0)$
$T(0) = t_c$

Algorytm:

```
int silnia(int n) {
    if (n == 0) return 1;
    else return (n * silnia(n – 1));
}
```

Parametr:
t_c – czas wykonania operacji dominującej.

Po przekształceniu:

$$T(n) + T(n-1) + \ldots + T(0) = (n+1)\,t_c + T(n-1) + \ldots + T(0)$$

Po uproszczeniu:

$$T(n) = (n+1)\,t_c$$

Wyznaczona w ten sposób funkcja T to złożoność praktyczna algorytmu.

Złożoność obliczeniowa teoretyczna

Złożoność obliczeniowa praktyczna:

- wyraża się względem parametru *n*,
- zmiana *n* na *n+1*, *n+10*, *n+20* itp., nie wpływa zasadniczo na *n* – obliczenia nadal zależą od nieznanego *n*,
- zmiana *n* **o rząd wielkości** ma wpływ na liczbę obliczeń.

Złożoność obliczeniowa praktyczna:

- wyraża się względem parametru *n*,
- zmiana *n* na *n+1*, *n+10*, *n+20* itp., nie wpływa zasadniczo na *n* – obliczenia nadal zależą od nieznanego *n*,
- zmiana *n* **o rząd wielkości** ma wpływ na liczbę obliczeń.

Złożoność teoretyczną można uprościć, pomijając parametry które nie zwiększają radykalnie liczby operacji.

Złożoność obliczeniowa praktyczna:

- wyraża się względem parametru *n*,
- zmiana *n* na *n+1*, *n+10*, *n+20* itp., nie wpływa zasadniczo na *n* – obliczenia nadal zależą od nieznanego *n*,
- zmiana *n* **o rząd wielkości** ma wpływ na liczbę obliczeń.

Złożoność teoretyczną można uprościć, pomijając parametry które nie zwiększają radykalnie liczby operacji.

Złożoność obliczeniowa teoretyczna – jest to uproszczenie funkcji T(*n*) określającej złożoność obliczeniową praktyczną.

Złożoność obliczeniowa praktyczna:

- wyraża się względem parametru *n*,
- zmiana *n* na *n+1*, *n+10*, *n+20* itp., nie wpływa zasadniczo na *n* – obliczenia nadal zależą od nieznanego *n*,
- zmiana *n* **o rząd wielkości** ma wpływ na liczbę obliczeń.

Złożoność teoretyczną można uprościć, pomijając parametry które nie zwiększają radykalnie liczby operacji.

Złożoność obliczeniowa teoretyczna – jest to uproszczenie funkcji T(*n*) określającej złożoność obliczeniową praktyczną.

Zapis formalny:

$$O(T(n)) = \left\{ g : T : N \to R^{*} \mid (\exists M \in R^{+})(\exists n_0 \in N)(\forall n \geqslant n_0) \right\} \left[|g(n)| \leqslant |M \cdot T(n)| \right]$$

gdzie: *R* to zbiór liczb rzeczywistych, *N* – naturalnych.

Złożoność obliczeniowa praktyczna:

- wyraża się względem parametru *n*,
- zmiana *n* na *n+1*, *n+10*, *n+20* itp., nie wpływa zasadniczo na *n* – obliczenia nadal zależą od nieznanego *n*,
- zmiana *n* **o rząd wielkości** ma wpływ na liczbę obliczeń.

Złożoność teoretyczną można uprościć, pomijając parametry które nie zwiększają radykalnie liczby operacji.

Złożoność obliczeniowa teoretyczna – jest to uproszczenie funkcji T(*n*) określającej złożoność obliczeniową praktyczną.

Zapis formalny:

$$O(T(n)) = \left\{ g : T : N \to R^{*} \mid (\exists M \in R^{+})(\exists n_0 \in N)(\forall n \geqslant n_0) \right\} \left[|g(n)| \leqslant |M \cdot T(n)| \right]$$

gdzie: *R* to zbiór liczb rzeczywistych, *N* – naturalnych.

Złożoność obliczeniowa teoretyczna = klasa algorytmu.

Reguły upraszczania złożoności praktycznej do teoretycznej:

- $c \cdot O(f(n)) = O(f(n))$ - usunięcie stałej sprzed funkcji,
- $O(f(n)) + O(f(n)) = O(f(n))$ - zapis równoważny $2 \cdot O(f(n))$
- $O(O(f(n))) = O(f(n))$
- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$
- $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n))$

Reguły upraszczania złożoności praktycznej do teoretycznej:

- $c \cdot O(f(n)) = O(f(n))$ - usunięcie stałej sprzed funkcji,
- $O(f(n)) + O(f(n)) = O(f(n))$ - zapis równoważny $2 \cdot O(f(n))$
- $O(O(f(n))) = O(f(n))$
- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$
- $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n))$

Złożoności praktyczne i odpowiadające im złożoności teoretyczne:

Złożoność praktyczna	**Złożoność teoretyczna**
T(n)	O(n)
T(3)	O(1)
T(n^2-n-1)	O(n^2)
T(2n+2)	O(n)
T(2^n+n^3+2)	O(2^n)

Klasy złożoności obliczeniowej

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Klasy złożoności obliczeniowej:

- O(1), → Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.

Funkcja, gdzie przyrost czasu będzie logarytmiczny, np. przy zwiększeniu rzędu danych o 10 przyrost czasu wykonania zwiększy się dwukrotnie (rząd danych rośnie geometrycznie 1,10,100,1000, a czas arytmetycznie 1,2,3,4 …).

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.

Funkcja, gdzie przyrost czasu będzie logarytmiczny, np. przy zwiększeniu rzędu danych o 10 przyrost czasu wykonania zwiększy się dwukrotnie (rząd danych rośnie geometrycznie 1,10,100,1000, a czas arytmetycznie 1,2,3,4 …).

Czas wykonywania algorytmu jest liniowo proporcjonalny do ilości przetwarzanych danych.

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.

Funkcja, gdzie przyrost czasu będzie logarytmiczny, np. przy zwiększeniu rzędu danych o 10 przyrost czasu wykonania zwiększy się dwukrotnie (rząd danych rośnie geometrycznie 1,10,100,1000, a czas arytmetycznie 1,2,3,4 …).

Czas wykonywania algorytmu jest liniowo proporcjonalny do ilości przetwarzanych danych.

Dla małych *k* jest to klasa wykładnicza, pojawiająca się w różnych operacjach matematycznych, operacjach na macierzach, itp.

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n).

Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.

Funkcja, gdzie przyrost czasu będzie logarytmiczny, np. przy zwiększeniu rzędu danych o 10 przyrost czasu wykonania zwiększy się dwukrotnie (rząd danych rośnie geometrycznie 1,10,100,1000, a czas arytmetycznie 1,2,3,4 …).

Czas wykonywania algorytmu jest liniowo proporcjonalny do ilości przetwarzanych danych.

Dla małych *k* jest to klasa wykładnicza, pojawiająca się w różnych operacjach matematycznych, operacjach na macierzach, itp.

W praktyce tego typu algorytmy stosuje się dla małych *n*.

Klasy złożoności obliczeniowej:

- O(1),
- O(log *n*),
- O(*n*),
- O(*nk*),
- O(2n),
- ...

Liczba występujących operacji jest niezależna od wielkości wprowadzanych danych. Na przykład: w pewnych funkcjach hashujących.

Funkcja, gdzie przyrost czasu będzie logarytmiczny, np. przy zwiększeniu rzędu danych o 10 przyrost czasu wykonania zwiększy się dwukrotnie (rząd danych rośnie geometrycznie 1,10,100,1000, a czas arytmetycznie 1,2,3,4 …).

Czas wykonywania algorytmu jest liniowo proporcjonalny do ilości przetwarzanych danych.

Dla małych *k* jest to klasa wykładnicza, pojawiająca się w różnych operacjach matematycznych, operacjach na macierzach, itp.

W praktyce tego typu algorytmy stosuje się dla małych *n*.

Można definiować inne typy złożoności, w praktyce jednak sprowadza się je do kilku typowych klas złożoności.

Typowe funkcje złożoności algorytmów:

Funkcja	**Złożoność**
1	– złożoność stała równa 1,
log(n)	– logarytmiczna,
n	– liniowa,
n log(n)	– liniowo-logarytmiczna,
n^2	– wielomianowa typu P,
n^3	– złożoności typu P,
n^4	– złożoności typu P,
2^n ($K^{W(n)}$)	– wykładnicza typy NP,
$n!$	– złożoność typu silnia,
n^n	– złożoność typu n^n.
...	

Typowe funkcje złożoności algorytmów:

Funkcja	**Złożoność**
1	– złożoność stała równa 1,
log(n)	– logarytmiczna,
n	– liniowa,
n log(n)	– liniowo-logarytmiczna,
n^2	– wielomianowa typu P,
n^3	– złożoności typu P,
n^4	– złożoności typu P,
2^n ($K^{W(n)}$)	– wykładnicza typy NP,
$n!$	– złożoność typu silnia,
n^n	– złożoność typu n^n.
...	

Granica algorytmu „dobrej jakości”

Typowe funkcje złożoności algorytmów:

Funkcja	**Złożoność**
1	– złożoność stała równa 1,
log(*n*)	– logarytmiczna,
n	– liniowa,
n log(*n*)	– liniowo-logarytmiczna,
n^2	– wielomianowa typu P,
n^3	– złożoności typu P,
n^4	– złożoności typu P,
2^n ($K^{W(n)}$)	– wykładnicza typy NP,
n!	– złożoność typu silnia,
n^n	– złożoność typu n^n.
...	

Granicą algorytmu „dobrej jakości”

Uwagi:

- Algorytmów **NP** lub o złożoności typu **silnia** praktycznie nie można skalować i tylko dla bardzo małej ilości danych można uzyskać wyniki w akceptowalnym czasie.

Typowe funkcje złożoności algorytmów:

Funkcja	**Złożoność**
1	– złożoność stała równa 1,
log(*n*)	– logarytmiczna,
n	– liniowa,
n log(*n*)	– liniowo-logarytmiczna,
n^2	– wielomianowa typu P,
n^3	– złożoności typu P,
n^4	– złożoności typu P,
2^n ($K^{W(n)}$)	– wykładnicza typy NP,
n!	– złożoność typu silnia,
n^n	– złożoność typu n^n.
...	

Granicą algorytmu „dobrej jakości”

Uwagi:

- Algorytmów **NP** lub o złożoności typu **silnia** praktycznie nie można skalować i tylko dla bardzo małej ilości danych można uzyskać wyniki w akceptowalnym czasie.
- Rozwiązania zadań, które wskazują na złożoność 2^n są trudne do wykonania i są nazywane **problemami NP** (np. problem NP-zupełny komiwojażera).

Typy złożoności obliczeniowej

Rozważmy funkcję wyszukującą element w tablicy:

```
function szukaj(tab: array[1..n] of Integer; X: Integer): Interer;
var
   pos: Integer;
begin
   while (pos < n) and (tab[pos] <> X do pos := pos + 1;
   if pos < n then szukaj := pos
   else szukaj := -1;
end;
```

Rozważmy funkcję wyszukującą element w tablicy:

```
function szukaj(tab: array[1..n] of Integer; X: Integer): Interer;
var
   pos: Integer;
begin
   while (pos < n) and (tab[pos] <> X do pos := pos + 1;
   if pos < n then szukaj := pos
   else szukaj := -1;
end;
```

Zadanie:

Oszacować złożoność obliczeniową algorytmu ?

Rozważmy funkcję wyszukującą element w tablicy:

```
function szukaj(tab: array[1..n] of Integer; X: Integer): Interer;
var
   pos: Integer;
begin
   while (pos < n) and (tab[pos] <> X do pos := pos + 1;
   if pos < n then szukaj := pos
   else szukaj := -1;
end;
```

Zadanie:

Oszacować złożoność obliczeniową algorytmu ?

Odpowiedź:

Nie można udzielić jednoznacznej odpowiedzi.

Rozważmy funkcję wyszukującą element w tablicy:

```
function szukaj(tab: array[1..n] of Integer; X: Integer): Interer;
var
   pos: Integer;
begin
   while (pos < n) and (tab[pos] <> X do pos := pos + 1;
   if pos < n then szukaj := pos
   else szukaj := -1;
end;
```

Zadanie:

Oszacować złożoność obliczeniową algorytmu ?

Odpowiedź:

Nie można udzielić jednoznacznej odpowiedzi.

Element można równie dobrze znaleźć na początku tablicy jak i na jej końcu.

Jeśli nie można ocenić jaka jest złożoność algorytmu należy określić typ złożoności.

Typy złożoności obliczeniowej:

- złożoność pesymistyczną,
- złożoność optymistyczną,
- złożoność średnią.

Jeśli nie można ocenić jaka jest złożoność algorytmu należy określić typ złożoności.

Typy złożoności obliczeniowej:

- złożoność pesymistyczną, → Jest to najgorszy możliwy przypadek danych wejściowych, gdzie algorytm wykonuje się najdłużej, posiada najwięcej operacji dominujących, zabiera najwięcej pamięci, wykorzystuje najwięcej wątków itp.
- złożoność optymistyczną,
- złożoność średnią.

Jeśli nie można ocenić jaka jest złożoność algorytmu należy określić typ złożoności.

Typy złożoności obliczeniowej:

- złożoność pesymistyczną,
- złożoność optymistyczną,
- złożoność średnią.

złożoność pesymistyczną → Jest to najgorszy możliwy przypadek danych wejściowych, gdzie algorytm wykonuje się najdłużej, posiada najwięcej operacji dominujących, zabiera najwięcej pamięci, wykorzystuje najwięcej wątków itp.

złożoność optymistyczną → Przypadek, gdzie dane wejściowe są tak dobrane, że występuje najmniejsza ilość operacji dominujących,

Jeśli nie można ocenić jaka jest złożoność algorytmu należy określić typ złożoności.

Typy złożoności obliczeniowej:

- złożoność pesymistyczną,
- złożoność optymistyczną,
- złożoność średnią.

Jest to najgorszy możliwy przypadek danych wejściowych, gdzie algorytm wykonuje się najdłużej, posiada najwięcej operacji dominujących, zabiera najwięcej pamięci, wykorzystuje najwięcej wątków itp.

Przypadek, gdzie dane wejściowe są tak dobrane, że występuje najmniejsza ilość operacji dominujących,

Przypadek, gdy dane wejściowe nie wymagają wykonywania dużej liczby operacji dominujących, jednocześnie nie ich liczba nie odpowiada złożoności optymistycznej.

Jeśli nie można ocenić jaka jest złożoność algorytmu należy określić typ złożoności.

Typy złożoności obliczeniowej:

- złożoność pesymistyczną,
- złożoność optymistyczną,
- złożoność średnią.

Jest to najgorszy możliwy przypadek danych wejściowych, gdzie algorytm wykonuje się najdłużej, posiada najwięcej operacji dominujących, zabiera najwięcej pamięci, wykorzystuje najwięcej wątków itp.

Przypadek, gdzie dane wejściowe są tak dobrane, że występuje najmniejsza ilość operacji dominujących,

Przypadek, gdy dane wejściowe nie wymagają wykonywania dużej liczby operacji dominujących, jednocześnie nie ich liczba nie odpowiada złożoności optymistycznej.

Uwagi:

- Dla różnych algorytmów pojęcie „**dobroci danych**” lub „**złośliwości danych**” jest uwarunkowane specyfiką wykonywanych przez algorytm obliczeń.
- Z punktu widzenia realizacji algorytmu najistotniejsza jest złożoność pesymistyczna, informująca o możliwym maksymalnym czasie wykonania zadania.

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.

$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.
$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Prawdopodobieństwo *p* pojawienia się danej *x* na *i*-tej pozycji: $P(D_{n,i}) = \frac{p}{n}$

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.
$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Prawdopodobieństwo *p* pojawienia się danej *x* na *i*-tej pozycji: $P(D_{n,i}) = \frac{p}{n}$

Prawdopodobieństwo nie pojawienia się szukanego elementu w tablicy: $P(D_{n,n}) = 1 - p$

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.
$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Prawdopodobieństwo *p* pojawienia się danej *x* na *i*-tej pozycji: $P(D_{n,i}) = \frac{p}{n}$

Prawdopodobieństwo nie pojawienia się szukanego elementu w tablicy: $P(D_{n,n}) = 1 - p$

Złożoności praktyczne wynoszą: $T(D_{n,i}) = i$ oraz $T(Dn,n) = n$, wtedy:

$$T_{śr} = \sum_{i=0}^{n} P(D_{n,i}) \cdot T(D_{n,i}) = (1-p)n + \sum_{i=0}^{n-1} i \frac{p}{n} = (1-p)n + (n+1)\frac{p}{2}$$

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.
$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Prawdopodobieństwo *p* pojawienia się danej *x* na *i*-tej pozycji: $P(D_{n,i}) = \frac{p}{n}$

Prawdopodobieństwo nie pojawienia się szukanego elementu w tablicy: $P(D_{n,n}) = 1 - p$

Złożoności praktyczne wynoszą: $T(D_{n,i}) = i$ oraz $T(Dn,n) = n$, wtedy:

$$T_{śr} = \sum_{i=0}^{n} P(D_{n,i}) \cdot T(D_{n,i}) = (1-p)n + \sum_{i=0}^{n-1} i \frac{p}{n} = (1-p)n + (n+1)\frac{p}{2}$$

Jeśli jest pewne, że szukana wartość znajduje się w tablicy, wtedy p = 1 i $T_{śr} = \frac{n+1}{2}$

Dla funkcji wyszukujące dane w tablicy:

- najgorszy przypadek jest równy rozmiarowi tablicy, $T_{pes}(n) = n$,
- złożoność optymistyczna wynosi $T_{opt}(n) = 1$,
- wyznaczenie złożoności średniej wymaga zastosowania rachunku prawdopodobieństwa:

$D_{i,n}$ – prawdopodobieństwo znalezienia wartości na *i*-tym miejscu z *n* możliwych.
$D_{n,n}$ – prawdopodobieństwo dla wartości nie występującej w tablicy.

Prawdopodobieństwo *p* pojawienia się danej *x* na *i*-tej pozycji: $P(D_{n,i}) = \frac{p}{n}$

Prawdopodobieństwo nie pojawienia się szukanego elementu w tablicy: $P(D_{n,n}) = 1 - p$

Złożoności praktyczne wynoszą: $T(D_{n,i}) = i$ oraz $T(Dn,n) = n$, wtedy:

$$T_{śr} = \sum_{i=0}^{n} P(D_{n,i}) \cdot T(D_{n,i}) = (1-p)n + \sum_{i=0}^{n-1} i \frac{p}{n} = (1-p)n + (n+1)\frac{p}{2}$$

Jeśli jest pewne, że szukana wartość znajduje się w tablicy, wtedy p = 1 i $T_{śr} = \frac{n+1}{2}$

Złożoności wynoszą: optymistyczna – O(1), średnia i pesymistyczna – O(*n*).

Uwagi:

- złożoność obliczeniowa nie jest jedynym wyznacznikiem jakości algorytmu,
- niektóre klasy wydajności – np. klasy wielomianowe *n* – dla małych porcji danych mogą charakteryzować się mniejszą liczbą operacji dominujących, niż klasy mniej złożone obliczeniowo,
- w każdym aspekcie programowania należy zachować trzeźwość umysłu i posługiwać się narzędziami w sposób odpowiedni i optymalny.

Problemy klasy P i NP

Problem klasy P – ang. deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można znaleźć w czasie wielomianowym.

Problem klasy P – ang. deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można znaleźć w czasie wielomianowym.

Problem klasy NP – ang. non-deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można zweryfikować w czasie wielomianowym.

Problem klasy P – ang. deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można znaleźć w czasie wielomianowym.

Problem klasy NP – ang. non-deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można zweryfikować w czasie wielomianowym.

Różnice między problemami P i NP:

- dla problemów P znalezienie rozwiązania ma złożoność wielomianową,
- dla problemów NP sprawdzenie gotowego rozwiązania można wykonać w czasie wielomianowym.

Problem klasy P – ang. deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można znaleźć w czasie wielomianowym.

Problem klasy NP – ang. non-deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można zweryfikować w czasie wielomianowym.

Różnice między problemami P i NP:

- dla problemów P znalezienie rozwiązania ma złożoność wielomianową,
- dla problemów NP sprawdzenie gotowego rozwiązania można wykonać w czasie wielomianowym.

Wszystkie problemy klasy P są także problemami klasy NP

Problem klasy P – ang. deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można znaleźć w czasie wielomianowym.

Problem klasy NP – ang. non-deterministic polynomial, to **problem decyzyjny**, którego rozwiązanie można zweryfikować w czasie wielomianowym.

Różnice między problemami P i NP:

- dla problemów P znalezienie rozwiązania ma złożoność wielomianową,
- dla problemów NP sprawdzenie gotowego rozwiązania można wykonać w czasie wielomianowym.

Wszystkie problemy klasy P są także problemami klasy NP

Nie wiadomo czy istnieje problem NP nie będący w klasie problemów P
(zagadnienie to należy do klasy problemów milenijnych)

Problemy klasy NP:

- **Problem NP-zupełny:**
  Jest to problem decyzyjny Q, należący do klasy NP, dla którego można wskazać inny problem Q' również należący do klasy NP, dla których redukcja Q' do Q możliwa jest w czasie wielomianowym.

Problemy klasy NP:

- **Problem NP-zupełny:**
  Jest to problem decyzyjny Q, należący do klasy NP, dla którego można wskazać inny problem Q' również należący do klasy NP, dla których redukcja Q' do Q możliwa jest w czasie wielomianowym.

Przykłady problemów NP-zupełnych:
- problem spełnialności formuł zdaniowych (SAT),
- cykl Hamiltona,
- problem komiwojażera,
- problem podziału grafu na kliki,
- problem kolorowania grafu.

Problemy klasy NP:

- **Problem NP-zupełny:**
  Jest to problem decyzyjny Q, należący do klasy NP, dla którego można wskazać inny problem Q' również należący do klasy NP, dla których redukcja Q' do Q możliwa jest w czasie wielomianowym.

Przykłady problemów NP-zupełnych:
- problem spełnialności formuł zdaniowych (SAT),
- cykl Hamiltona,
- problem komiwojażera,
- problem podziału grafu na kliki,
- problem kolorowania grafu.

Zagadnienie problemów NP-zupełnych jest szczególnie istotne dla kryptografii, np. złamanie szyfrowania RSA wymaga rozkładu liczby na czynniki pierwsze, które dla dowolnej liczby nie jest problemem wielomianowym.

Problemy klasy NP:

- **Problem NP-trudny:**
  Jest to taki problem, którego rozwiązanie jest co najmniej tak trudne, jak rozwiązanie dowolnego problemu z klasy NP.

Problemy klasy NP:

- **Problem NP-trudny:**
  Jest to taki problem, którego rozwiązanie jest co najmniej tak trudne, jak rozwiązanie dowolnego problemu z klasy NP.

Przykłady problemów NP-trudnych:
- problem komiwojażera,
- problem plecakowy.

Problemy klasy NP:

- **Problem NP-trudny:**
  Jest to taki problem, którego rozwiązanie jest co najmniej tak trudne, jak rozwiązanie dowolnego problemu z klasy NP.

Przykłady problemów NP-trudnych:
- problem komiwojażera,
- problem plecakowy.

Uwagi:
- dowolny problem NP-zupełny może być rozwiązany problemem NP-trudnym,
- problemy NP-trudne można rozpatrywać w różnych klasach, np.:
  - decyzyjne,
  - optymalizacyjne,
  - przeszukujące,
  - konstrukcyjne.

Koniec wykładu