ALGORYTMY I STRUKTURY DANYCH
WYKŁAD VI (materiały pomocnicze)

Metoda „zachłanna”

Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 21 grudnia 2008

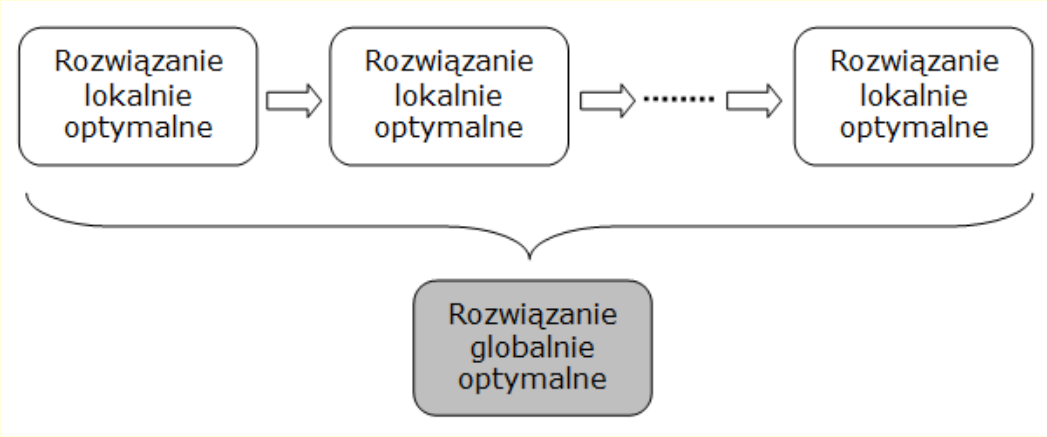
Plan wykładu:

- podstawy metody „zachłannej”,
- problem plecakowy:
  - w wersji ciągłej,
  - w wersji dyskretnej,
- algorytm Dijkstry,
- algorytm Kruskala,
- algorytm budowy drzewa kodu Huffmana,
- podejście „zachłanne” a algorytmy genetyczne.

Podstawy metody „zachłannej”

Podstawy metody „zachłannej”

W każdy kroku działania algorytmu, w którym trzeba dokonać wyboru, wybieramy zawsze rozwiązanie lokalnie optymalne, tj. takie, które w danej chwili działania algorytmu jest najlepsze względem pewnej miary. Przy „odrobinie szczęścia”, takie postępowanie prowadzi do rozwiązania globalnie optymalnego.

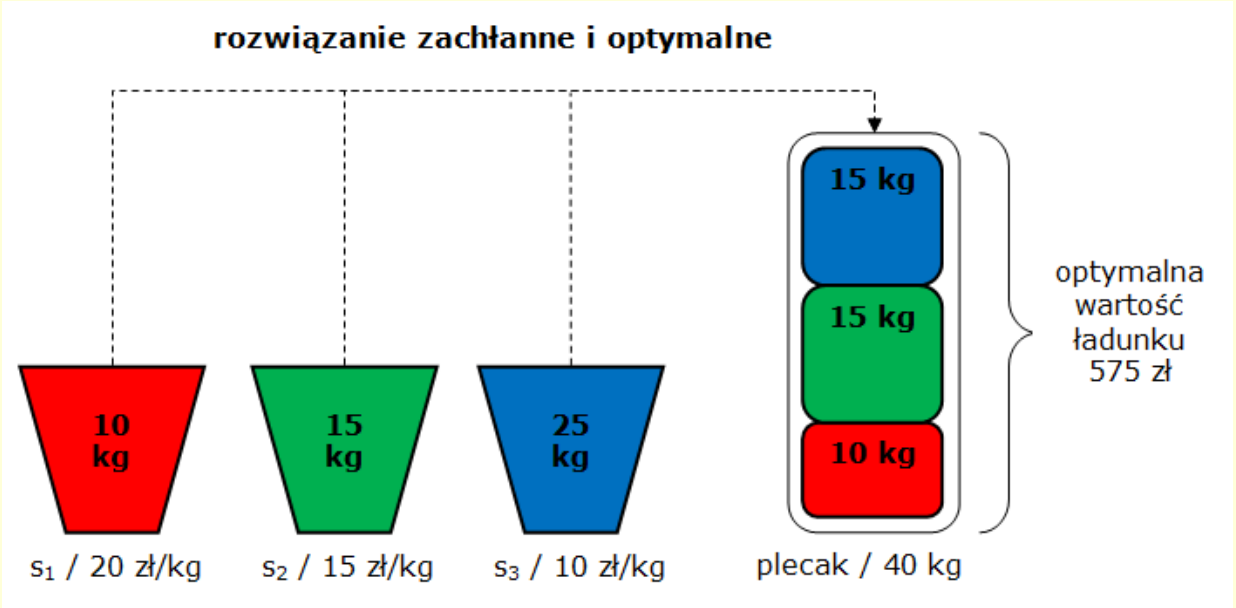


Problem plecakowy

Problem plecakowy – w wersji ciągłej

Szkic. W skarbcu pewnego banku znajduje się n skrzyń $s_1, s_2, \ldots, s_n$, z których każda zawiera odpowiednio $k_1, k_2, \ldots, k_n$ kilogramów rozdrobnionego kruszcu o wartości rynkowej $c_1, c_2, \ldots, c_n$ złotych za kilogram. Złodziej, który przedostał się do skarbca, zamierza wynieść w plecaku ładunek o jak największej wartości, jednak bezpieczna ucieczka wymaga od niego dużej sprawności, co ogranicza ilość przenoszonego ładunku do m kilogramów. Jak powinien zapakować swój plecak?

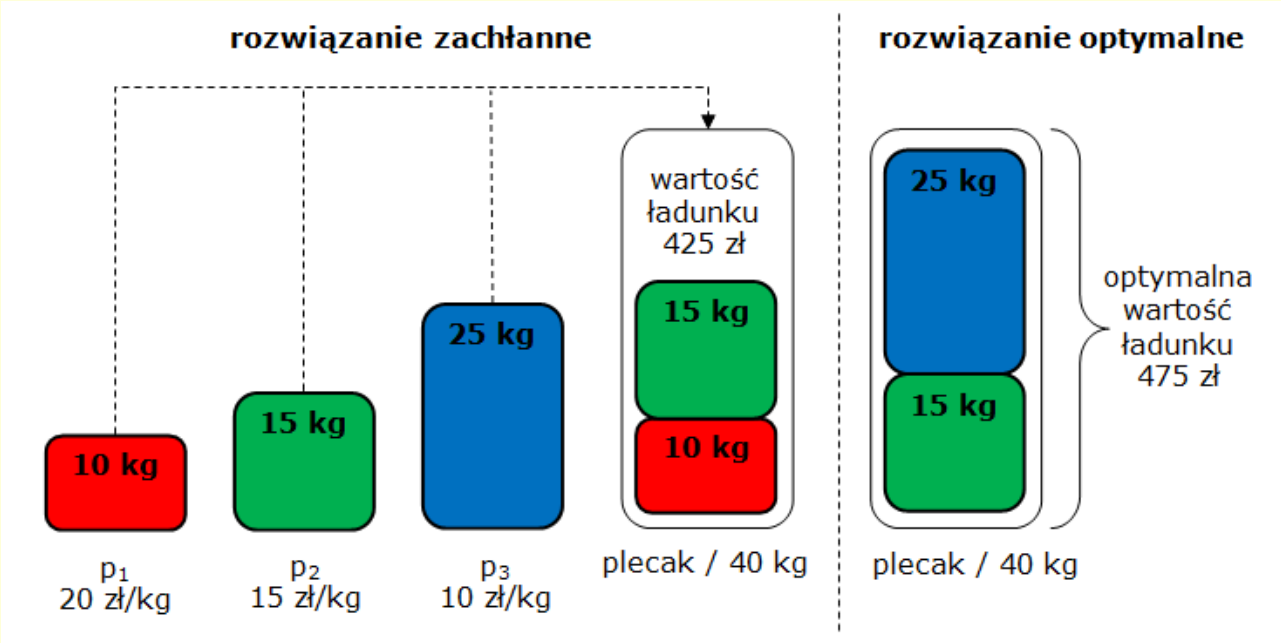
Przykład.



Problem plecakowy – w wersji dyskretnej

Szkic. W skarbcu pewnego banku znajduje się n przedmiotów $p_1, p_2, \ldots, p_n$, z których każdy waży odpowiednio $k_1, k_2, \ldots, k_n$ kilogramów, i których wartość rynkowa wynosi $c_1, c_2, \ldots, c_n$ złotych za kilogram. Złodziej, który przedostał się do skarbca, zamierza wynieść w plecaku ładunek o jak największej wartości, jednak bezpieczna ucieczka wymaga od niego dużej sprawności, co ogranicza ilość przenoszonego ładunku do m kilogramów. Jak powinien zapakować swój plecak?

Przykład.

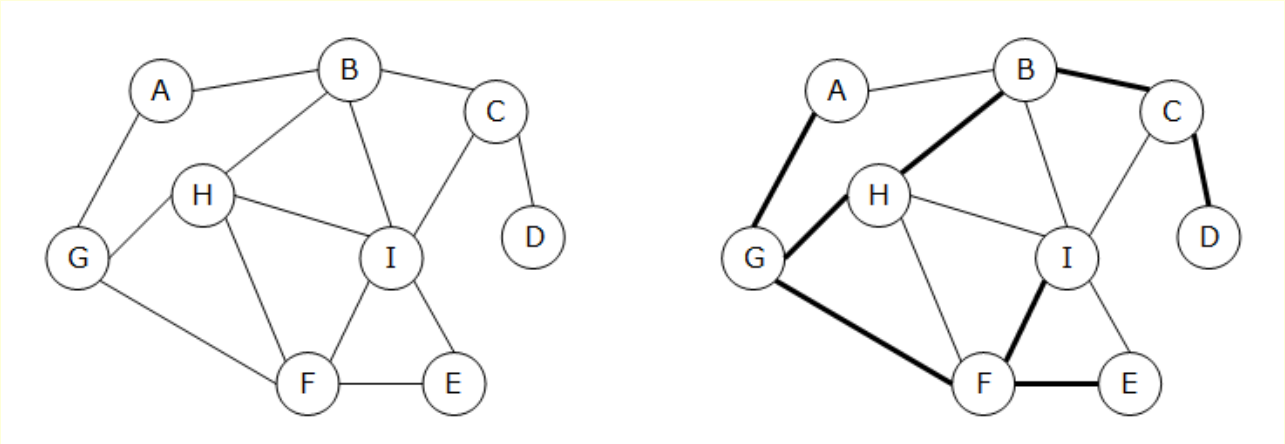


Algorytm Dijkstry

Algorytm Dijkstry

Definicja. *Drzewem rozpinającym* $ST(G)$ grafu nieskierowanego $G = (V, E)$ nazywamy podgraf-drzewo $G' = (V, E')$ grafu G taki, że dowolne dwa wierzchołki $u, v \in V$ są połączone drogą w grafie G'.

Przykład.

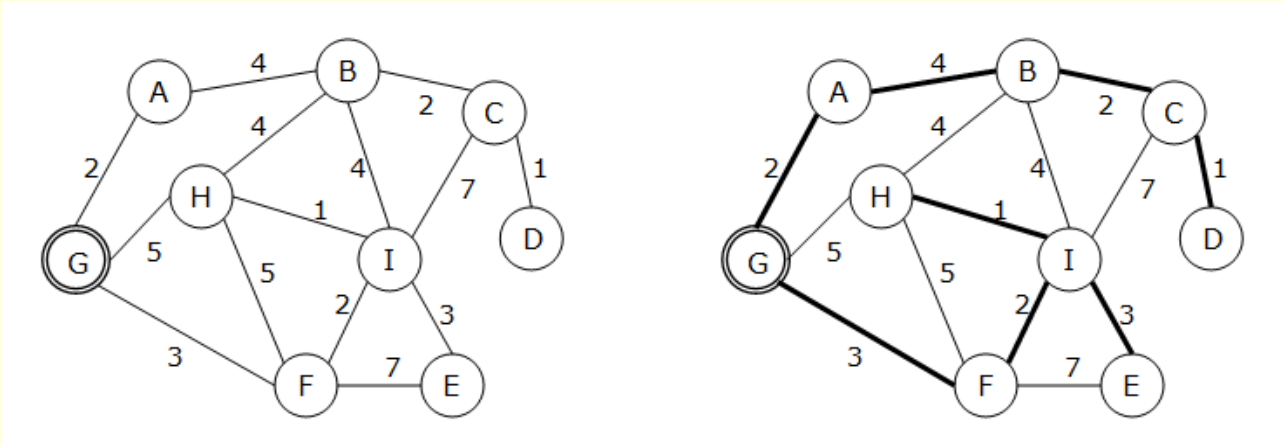


Algorytm Dijkstry

Definicja. *Drzewem rozpinającym „najkrótszych ścieżek" z wyróżnionego wierzchołka źródłowego* s nazywamy drzewo rozpinające $ST(G)$ grafu nieskierowanego z wagami $G = (V, E, f)$, gdzie $f : E \to \mathbb{N}^+$ takie, że dla każdego wierzchołka $u \in V$ i $u \neq s$, suma krawędzi na drodze z wierzchołka s do wierzchołka u w drzewie $ST(G)$ jest nie większa od sumy krawędzi na dowolnej z dróg łączących wierzchołki s i u w grafie G.

Problem. W zadanym grafie nieskierowanym z wagami $G = (V, E, f)$, gdzie $f : E \to \mathbb{N}^+$, wyznaczyć drzewo rozpinające najkrótszych ścieżek z wyróżnionego wierzchołka źródłowego $s \in V$.

Przykład.



Algorytm Dijkstry

Idea algorytmu Dijkstry. Zbiór wierzchołków grafu wejściowego $G = (V, E, f)$ dzielimy na trzy rozłączne podzbiory V_1, V_2, V_3 takie, że $V_1 \cup V_2 \cup V_3 = V$ oraz:

- V_1 – zbiór wierzchołków „zaakceptowanych", dla których ustaliliśmy już minimalną odległość od wierzchołka źródłowego s, początkowo $V_1 = \emptyset$,
- V_2 – zbiór wierzchołków „analizowanych", dla których aktualnie ustalamy odległość od wierzchołka źródłowego s, początkowo $V_2 = \{s\}$,
- V_3 – zbiór wierzchołków „nieanalizowanych", dla których jeszcze nie rozpoczęliśmy procesu ustalania odległości od wierzchołka źródłowego s, początkowo $V_3 = V \setminus \{s\}$.

Tworzymy dwie tablice pomocnicze rozmiaru $|V|$ kolejno $dist$ i $parent$, gdzie:

- $d[u]$ – długość aktualnie najkrótszej ścieżki z wierzchołka źródłowego s do wierzchołka u, początkowo $d[s] = 0$, w p.p. $d[v] = \infty$,
- $p[u]$ – poprzednik wierzchołka u na aktualnie najkrótszej ścieżce z wierzchołka źródłowego s do wierzchołka u, początkowo $p[s] = s$, w p.p. $p[v] = null$.

Algorytm Dijkstry

Idea algorytmu Dijkstry (c.d).

Dopóki zbiór V_2 nie jest pusty:

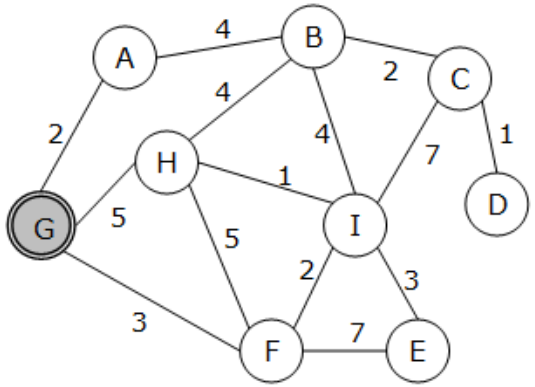
- wybieram ze zbioru V_2 wierzchołek u najbliższy wierzchołkowi źródłowemu s,
- dla każdego wierzchołka v sąsiedniego z wierzchołkiem u:
  - jeżeli $v \in V_2$, sprawdzamy, czy nowa droga z s do v przez u nie jest krótsza od dotychczasowej (ew. aktualizujemy $d[v]$ i $p[v]$),
  - jeżeli $v \in V_3$, wyznaczmy długość drogi z s do v przez u i wykonujemy $V_3 = V_3 \setminus \{v\}$ i $V_2 = V_2 \cup \{v\}$, aktualizujemy $d[v]$ i $p[v]$,
- wykonujemy $V_2 = V_2 \setminus \{u\}$ i $V_1 = V_1 \cup \{u\}$.

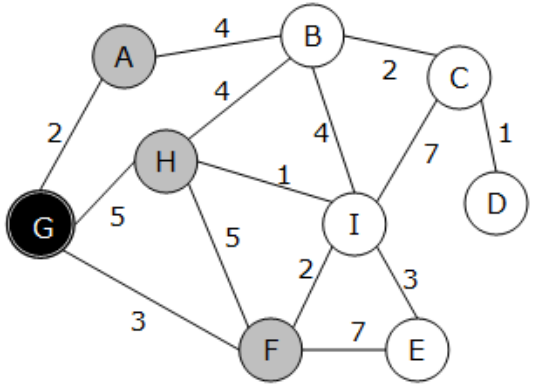
Drzewo rozpinające najkrótszych ścieżek z wyróżnionego wierzchołka źródłowego $s \in V$ odczytujemy z tablic d oraz p.

Pytanie. Jaka jest złożoność czasowa algorytmu Dijkstry dla grafu wejściowego $G = (V, E, f)$, gdzie $|V| = n$, jeżeli rozważym implementację zbioru V_2 w kolejce priorytetowej zapisanej w kopcu binarnym?

Algorytm Dijkstry

Przykład – pierwszy krok algorytmu.





A	B	C	D	E	F	G	H	I
-	-	-	-	-	-	0	-	-

|d|

A	B	C	D	E	F	G	H	I
2	-	-	-	-	3	0	5	-

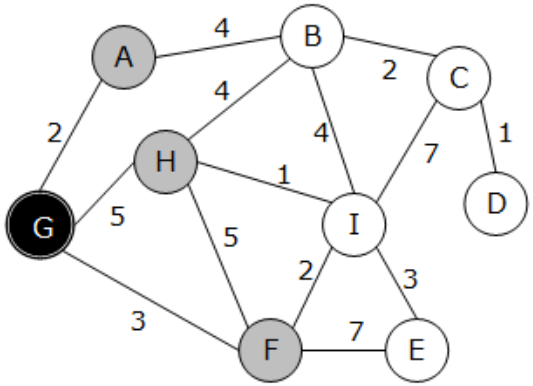
A	B	C	D	E	F	G	H	I
-	-	-	-	-	-	G	-	-

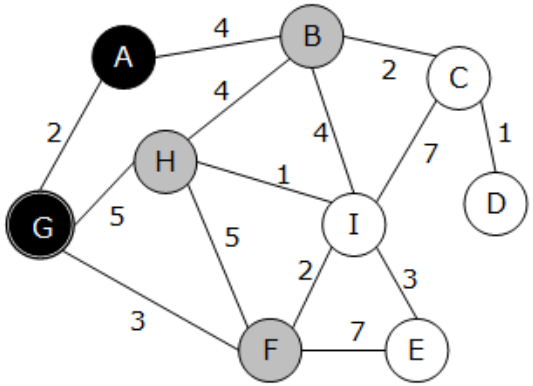
|p|

A	B	C	D	E	F	G	H	I
G	-	-	-	-	G	G	G	-

Algorytm Dijkstry

Przykład – drugi krok algorytmu.





A	B	C	D	E	F	G	H	I
2	-	-	-	-	3	0	5	-

|d|

A	B	C	D	E	F	G	H	I
2	6	-	-	-	3	0	5	-

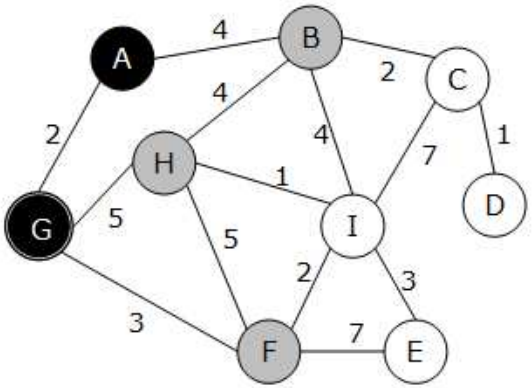
A	B	C	D	E	F	G	H	I
G	-	-	-	-	G	G	G	-

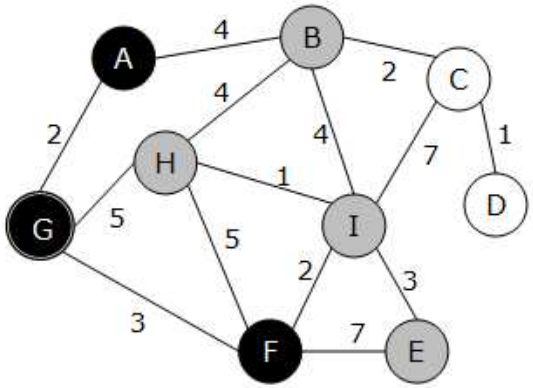
|p|

A	B	C	D	E	F	G	H	I
G	A	-	-	-	G	G	G	-

Algorytm Dijkstry

Przykład – trzeci krok algorytmu.





A	B	C	D	E	F	G	H	I
2	6	-	-	-	3	0	5	-

|d|

A	B	C	D	E	F	G	H	I
2	6	-	-	10	3	0	5	5

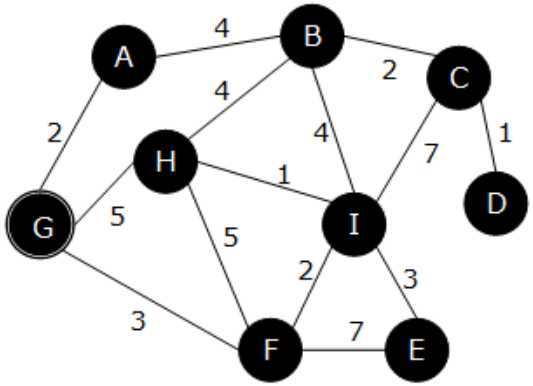
A	B	C	D	E	F	G	H	I
G	A	-	-	-	G	G	G	-

|p|

A	B	C	D	E	F	G	H	I
G	A	-	-	F	G	G	G	F

Algorytm Dijkstry

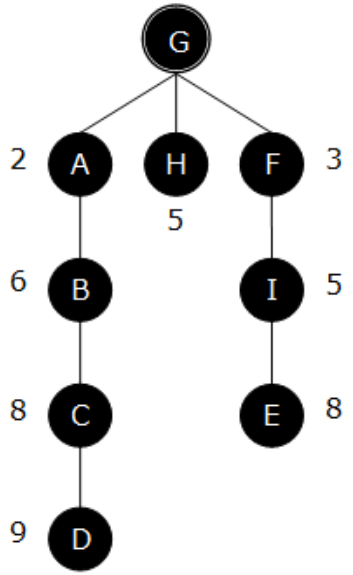
Przykład – ostatni krok algorytmu, rozwiązanie.



A	B	C	D	E	F	G	H	I	|d|
2	6	8	9	8	3	0	5	5	

A	B	C	D	E	F	G	H	I	|p|
G	A	B	C	I	G	G	G	F	

Drzewo najkrótszych ścieżek

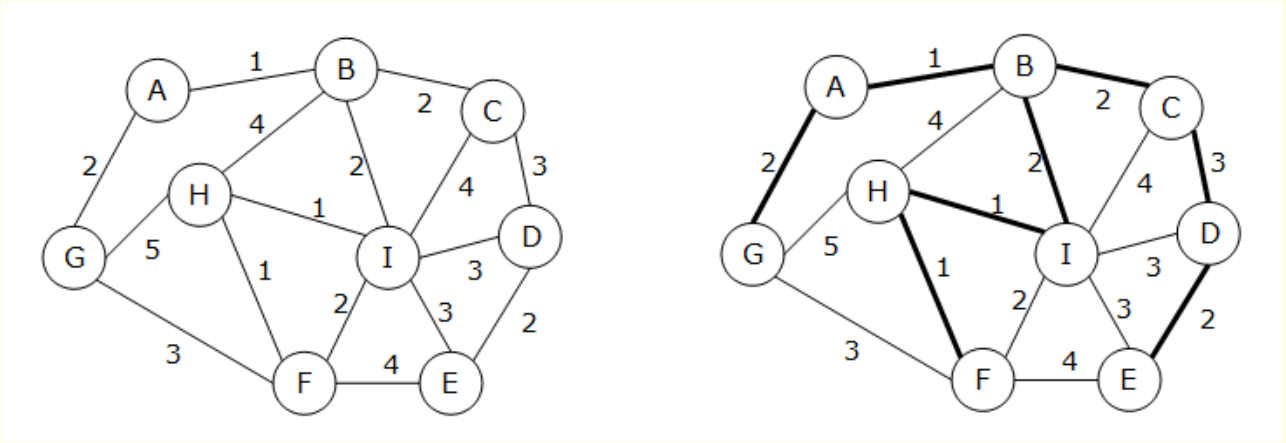


Algorytm Kruskala

Algorytm Kruskala

Definicja. *Minimalnym drzewem rozpinającym* $MST(G)$ grafu nieskierowanego z wagami $G = (V, E, f)$ nazywamy dowolne drzewo rozpinające tego grafu, którego suma krawędzi składowych jest nie większa od sumy krawędzi składowych każdego drzewa rozpinającego grafu G.

Przykład.



Algorytm Kruskala

Algorytm Kruskala. Niech $G = (V, E, f)$ będzie niezorientowanym grafem z wagami, gdzie $|E| = k$, $|V| = n$, oraz:

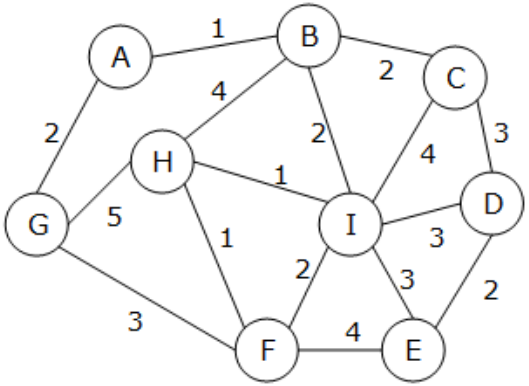
- pq – kolejka priorytetowa zawierająca wszystkie krawędzie grafu G zgodnie z porządkiem wag,
- U – początkowy podział zbioru wierzchołków grafu, tj. $U = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$,

wtedy

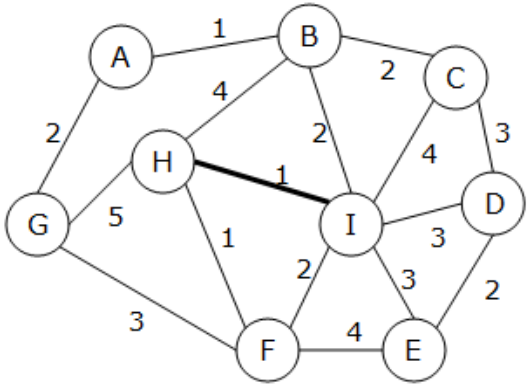
```
MinimumSpanningTree Kruskal(Graph g) {
   Edge e;
   MinimumSpanningTree MST;

   while (not empty(pq)) {
      e=min(pq);
      pq=delmin(pq);
      if (find(U,e.u)!=find(U,e.v)) {
         U=union(find(U,e.u),find(U,e.v));
         MST=Dodaj_krawędź(MST,e); }
   }
}
```

Algorytm Kruskala

Przykład – pierwszy krok algorytmu.



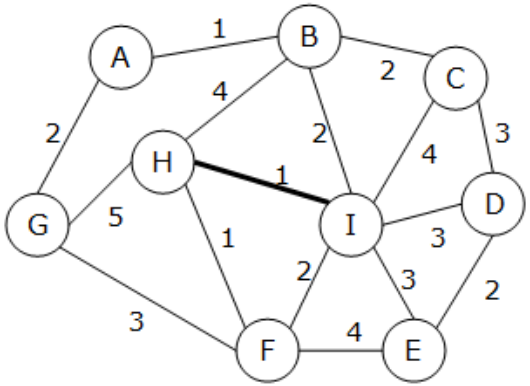
U={{A},{B},{C},{D},{E},
{F},{G},{H},{I}}



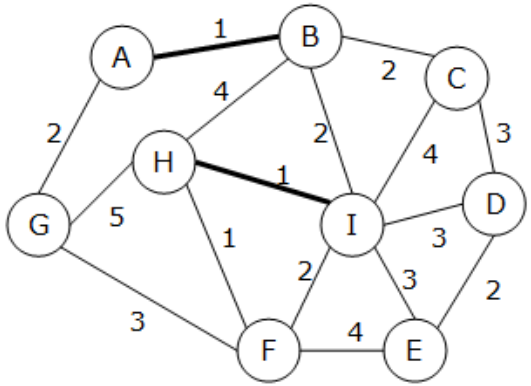
U={{A},{B},{C},{D},{E},
{F},{G},{H,I}}

Algorytm Kruskala

Przykład – drugi krok algorytmu.



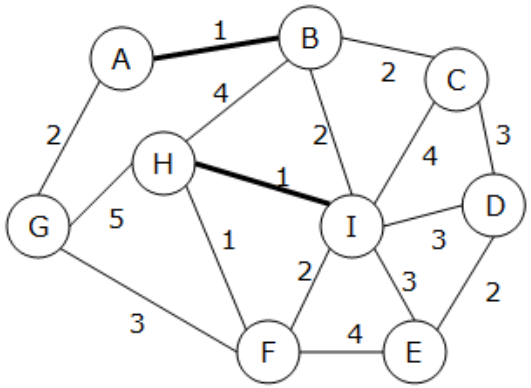
U={{A},{B},{C},{D},{E},
{F},{G},{H,I}}



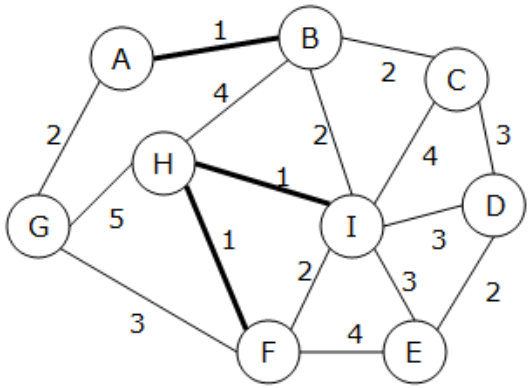
U={{A,B},{C},{D},{E},
{F},{G},{H,I}}

Algorytm Kruskala

Przykład – trzeci krok algorytmu.



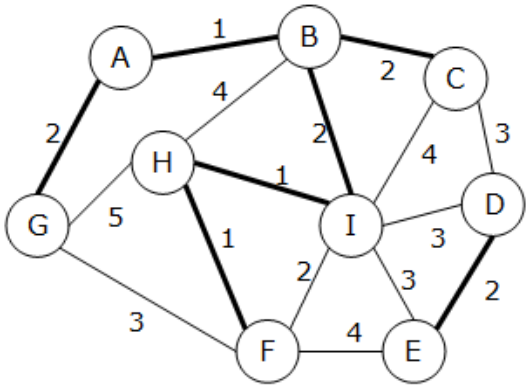
U={{A,B},{C},{D},{E},
{F},{G},{H,I}}



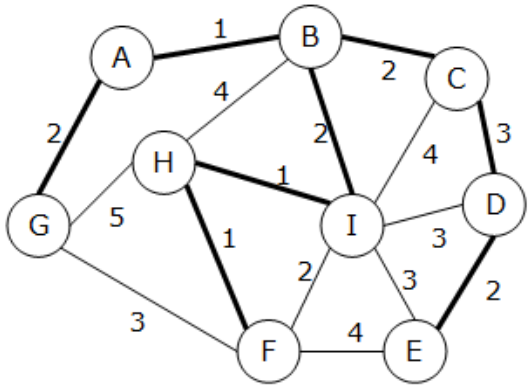
U={{A,B},{C},{D},{E},
{F,H,I},{G}}

Algorytm Kruskala

Przykład – ostatni krok algorytmu, rozwiązanie.



U={{A,B,C,F,G,H,I},{D,E}}



U={A,B,C,D,E,F,G,H,I}

Algorytm Kruskala

Złożoność rozwiązania. Niech $G = (V, E, f)$ będzie niezorientowanym grafem, gdzie $|E| = k$, $|V| = n$, gdzie:

- implementacja kolejki priorytetowej – kopiec typu min realizowany w tablicy,
- implementacja struktury Find-Union – drzewa n-arne z balansowaniem i kompresją ścieżek,

wtedy:

- koszt utworzenia kolejki priorytetowej – $O(k)$,
- koszt k operacji kolejki priorytetowej min – $k \cdot O(1) = O(k)$,
- koszt k operacji kolejki priorytetowej $delmin$ – $k \cdot O(\lg k) = O(k \lg k)$,
- koszt przemieszanych operacji struktury Find-Union, $2k + 2(n-1)$ operacji $find$ oraz $n-1$ operacji $union$ – $O((2k + 2(n-1) + n - 1) \lg^* (n-1)) = O(k \lg^* k)$.

Ostatecznie $O(k) + O(k) + O(k \lg k) + O(k \lg^* k) = O(k \lg k)$.

Algorytm budowy drzewa kodu Huffmana

Algorytm budowy drzewa kodu Huffmana

Problem. Niech Σ będzie alfabetem składającym się z n znaków. Dla danego tekstu Δ, zapisanego przy użyciu k znaków ze zbioru Σ, podać jednoznaczną reprezentację bitową o jak najmniejszej długości.

Rozwiązanie – stała długość słowa kodującego. Każdy znak alfabetu kodujemy różnowartościowo przy użyciu $\lceil \lg n \rceil$ bitów. Długość reprezentacji bitowej tekstu $k \cdot \lceil \lg n \rceil$.

Przykład. $\Sigma = \{a, b, c, d, e\}$, $\Delta = abbabacadeaaa$, wtedy jednoznaczny kod binarny stałej długości dla znaków alfabetu może mieć postać np.:

$$\begin{aligned} a &= 000, \\ b &= 001, \\ c &= 010, \\ d &= 011, \\ e &= 100. \end{aligned}$$

Zatem długość jednoznacznej reprezentacji bitowej tekstu Δ jest równa $13 \cdot 3 = 39$ bitów.

Algorytm budowy drzewa kodu Huffmana

Rozwiązanie – zmienna długość słowa kodującego. Dla każdego znaku alfabetu wyznaczamy częstość występowania w tekście Δ. Na tej podstawie dobieramy kod znaków zgodnie z regułą mówiącą, że:

- znakom występującym najczęściej przypisujemy najkrótsze słowo kodowe,
- znakom występującym najrzadziej przypisujemy najdłuższe słowo kodowe.

Przykład. $\Sigma = \{a, b, c, d, e\}$, $\Delta = abbabacadeaaa$, stąd $a - 7, b - 3, c - 1, d - 1, e - 1$, jednoznaczny kod binarny zmiennej długości dla znaków alfabetu może mieć postać np.:

$$\begin{aligned} a &= 0, \\ b &= 10, \\ c &= 110, \\ d &= 1110, \\ e &= 1111. \end{aligned}$$

Zatem długość jednoznacznej reprezentacji bitowej tekstu Δ jest równa $7 \cdot 1 + 3 \cdot 2 + 1 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 = 24$ bity.

Algorytm budowy drzewa kodu Huffmana

Definicja. Kod binarny o zmiennej długości słowa kodującego, dla alfabetu $\Sigma = \{e_1, e_2, \ldots, e_n\}$, nazywamy *kodem prefiksowym*, gdy kod binarny żadnego ze znaków e_i nie jest prefiksem kodu każdego ze znaków e_j, gdzie $1 \leq i, j \leq n$ i $i \neq j$.

Zadanie. Podaj przykład kodu binarnego o zmiennej długości dla alfabetu $\Sigma = \{a, b, c, d, e\}$, który:

- jest nie jest kodem prefiksowym,
- jest kodem prefiksowym.

Definicja. Prefiksowy kod binarny dla alfabetu Σ i tekstu kodowanego Δ nazywamy *kodem Huffmana*, gdy długość jednoznacznej reprezentacji bitowej tekstu Δ dla dowolnego innego kodu prefiksowego jest co najmniej równa długości jednoznacznej reprezentacji bitowej tekstu Δ dla tego kodu.

Twierdzenie. *Kod Huffmana jest optymalnym (pod względem długości jednoznacznej reprezentacji bitowej tekstu) kodem o zmiennej długości słowa kodowego dla ustalonego alfabetu Σ i tekstu kodowanego Δ.*

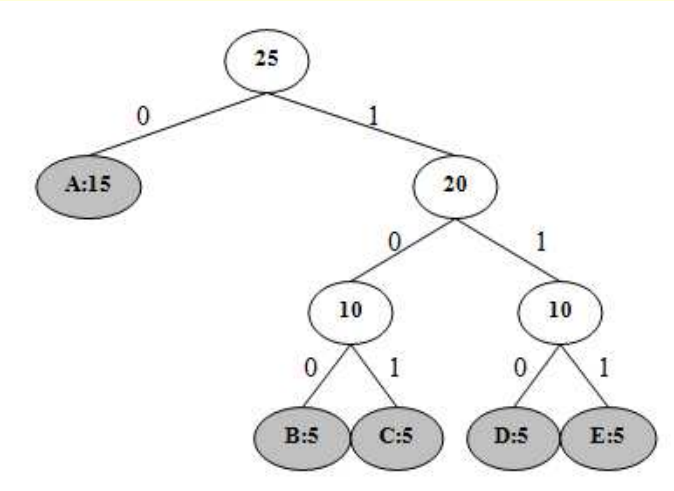
Algorytm budowy drzewa kodu Huffmana

Drzewo kodu Huffmana. Lokalnie pełne drzewo binarne dla ustalonego alfabetu Σ i tekstu kodowanego Δ, w którym:

- etykiety wierzchołków zewnętrznych reprezentują częstość występowania znaków w tekście,
- etykiety wierzchołków wewnętrznych odpowiadają sumie częstości występowania wszystkich znaków w tekście dla poddrzewa danego wierzchołka,
- dla każdego wierzchołka wewnętrznego etykieta jego lewego i prawego następnika jest odpowiednio nie większa i nie mniejsza od etykiety tego wierzchołka,
- krawędzie lewostronne i prawostronne drzewa etykietowane są odpowiednio 0 oraz 1.

Algorytm budowy drzewa kodu Huffmana

Przykład. Drzewo kodu Huffmana dla tekstu Δ zawierającego 15 znaków a oraz po 5 znaków b, c, d i e.



Algorytm budowy drzewa kodu Huffmana

Algorytm. Niech pq będzie kolejką priorytetową drzew kodu Huffmana. Początkowo kolejka zawiera las jednowierzchołkowych drzew reprezentujących znaki z alfabetu Σ i ich częstość występowania w ustalonym tekście Δ, wtedy:

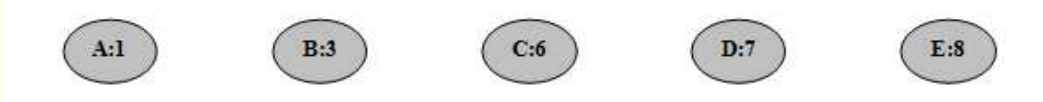
```
HTreeNode HuffmanTree(PriorityQueue pq) {
   HTreeNode x,y;

   x=min(pq); pq=delmin(pq);
   while (not empty(pq)) {
      y=min(pq); pq=delmin(pq);

      pq=insert(pq,połącz(x,y));

      x=min(pq); pq=delmin(pq);
   }
   return x;
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu HuffmanTree dla alfabetu składającego się z n znaków?

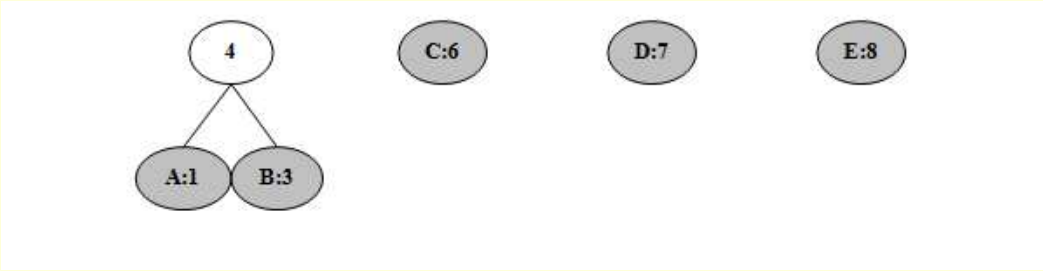
Algorytm budowy drzewa kodu Huffmana

Przykład. Budowa drzewa kodu Huffmana dla tekstu Δ zawierającego 1 znak a, 3 znaki b, 6 znaków c, 7 znaków d oraz 8 znaków e.

- Początkowy stan kolejki priorytetowej pq.



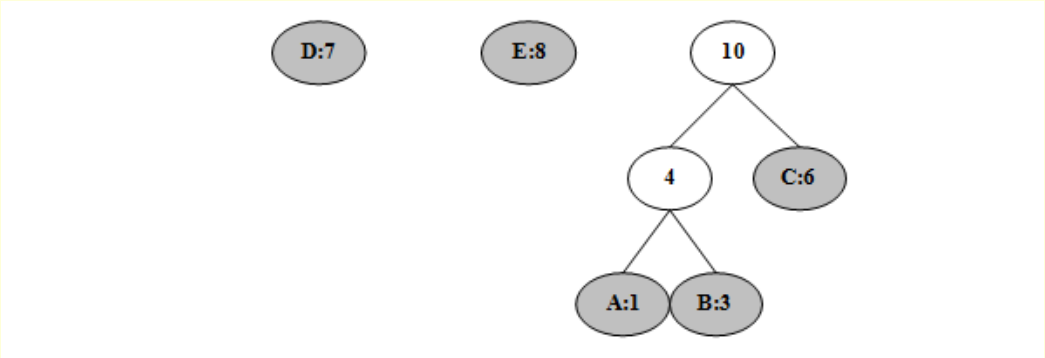
- Pierwszy krok algorytmu.



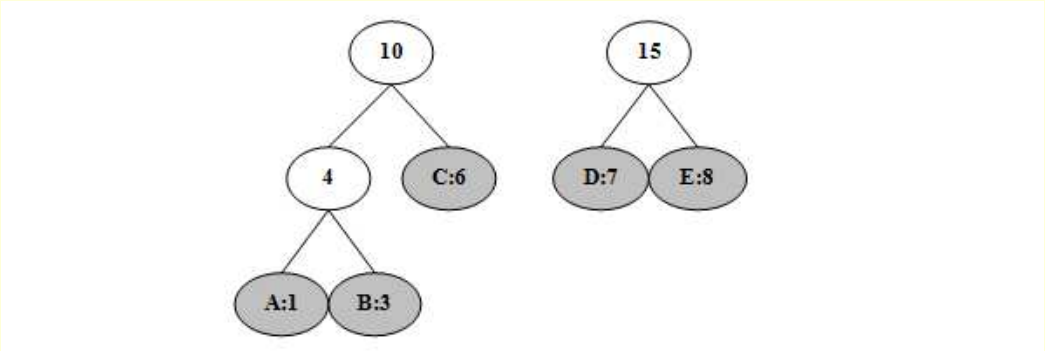
Algorytm budowy drzewa kodu Huffmana

Przykład (c.d.).

- Drugi krok algorytmu.



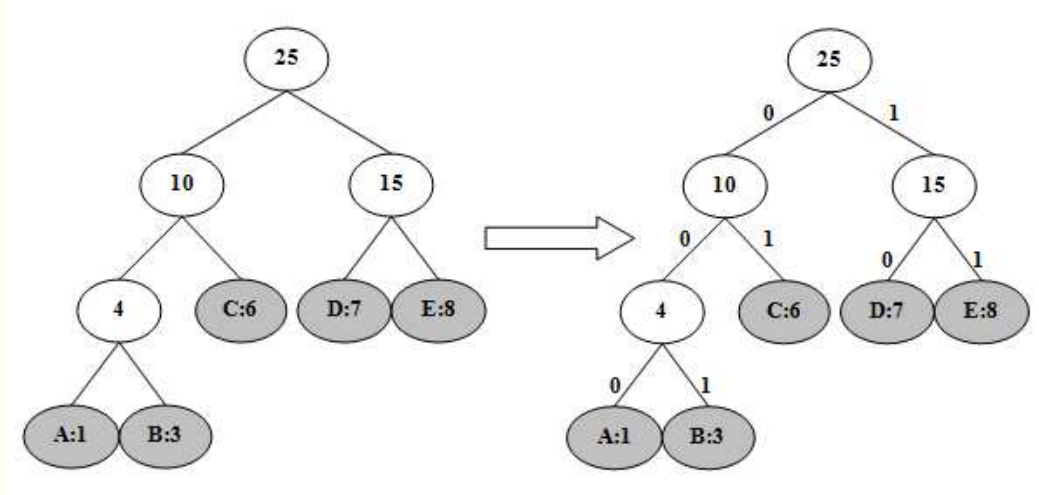
- Trzeci krok algorytmu.



Algorytm budowy drzewa kodu Huffmana

Przykład (c.d.).

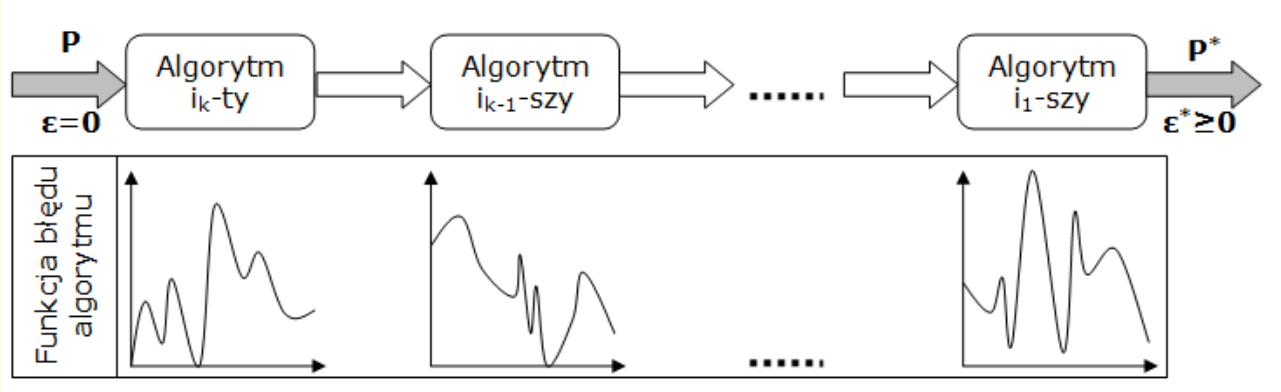
- Czwarty krok algorytmu i postać finalna drzewa kodu Huffmana.



Podejście „zachłanne” a algorytmy genetyczne

Podejście zachłanne a algorytmy genetyczne

Problem odkodowania obrazu. Dany jest zakodowany, k schematami zabezpieczeń z błędem początkowym $\epsilon = 0$, obraz cyfrowy P oraz k algorytmów $A_1, A_2, \ldots, A_k$ odkodowujących właściwy im schemat zabezpieczeń. Obraz P^* jest przybliżeniem obrazu źródłowego P z błędem $\epsilon^* \geq 0$, gdzie $P^* = A_{i_k}\left(A_{i_{k-1}}\left(\ldots A_{i_1}\left(P\right)\ldots\right)\right)$ oraz $i_k, i_{k-1}, \ldots, i_1$ będącego dowolną permutacją ciągu liczb $k, k-1, \ldots, 1$.Dla każdego z algorytmów A_i określono funkcję błędu odkodowania $f_i : \mathbb{R}^+ \cup \{0\} \rightarrow \mathbb{R}^+ \cup \{0\}$, której argumentem jest wstępny błąd odkodowania poprzedzający wykonanie algorytmu A_i. Podać permutację liczb $k, k-1, \ldots, 1$ taką, że ostateczny błąd ϵ^* odkodowania obrazu P, tj. błąd operacji $A_{i_k}\left(A_{i_{k-1}}\left(\ldots A_{i_1}\left(P\right)\ldots\right)\right)$ jest minimalny z możliwych.



Pytanie. Czy prosta strategia zachłanna jest poprawnym rozwiązaniem dla powyższego

problemu?

Podejście zachłanne a algorytmy genetyczne – kodowanie osobników

Definicja. *Przestrzenią rozwiązań* Δ nazywamy zbiór wszystkich możliwych odpowiedzi dla zadanego problemu.

Pytanie. Co jest przestrzenią rozwiązań dla problemu odkodowania obrazu?

Pytanie. Jaka jest moc przestrzeni rozwiązań dla problemu odkodowania obrazu?

Definicja. *Osobnikiem* nazywamy dowolny element α przestrzeni rozwiązań.

Definicja. *Chromosomem* $CH(\alpha)$ nazywam implementację osobnika $\alpha \in \Delta$.

Przykład. Dla zadania odkodowania obrazu przyjmiemy następującą postać chromosomu

$$CH(\alpha) = \left[\begin{array}{ccccccc} i_k & & i_{k-1} & & i_{k-2} & \ldots & i_2 \quad i_1 \end{array} \right],$$

gdzie $i_k, i_{k-1}, \ldots, i_1$ jest dowolną permutacją ciągu liczb $k, k-1, \ldots, 1$.

Definicja. *i-tym genem* $G(CH(\alpha), i)$ nazywam i-ty element chromosomu reprezentującego osobnika $\alpha \in \Delta$.

Podejście zachłanne a algorytmy genetyczne – proces selekcji

Definicja. *Funkcją przystosowania* $adapt : \Delta \rightarrow [0,1]$ nazywam wybraną funkcję determinującą przydatność osobnika w odniesieniu do kryteriów poszukiwanego rozwiązania.

Przykład. Dla zadania odkodowania obrazu przyjmiemy następującą postać funkcji przystosowania osobnika $\alpha \in \Delta$, dla $CH(\alpha) = \left[\begin{array}{ccccccc} i_k & & i_{k-1} & & i_{k-2} & \ldots & i_2 \quad i_1 \end{array} \right]$

$$adapt(\alpha) = \frac{1}{1+\epsilon^*},$$

gdzie ϵ^* jest błędem odkodowania obrazu P, dla operacji $A_{i_k}\left(A_{i_{k-1}}\left(\ldots A_{i_1}(P)\ldots\right)\right)$.

Definicja. *i-elementową populacją* nazywamy dowolny multizbiór $POP(\Delta, i)$, którego elementami są osobniki $\alpha \in \Delta$.

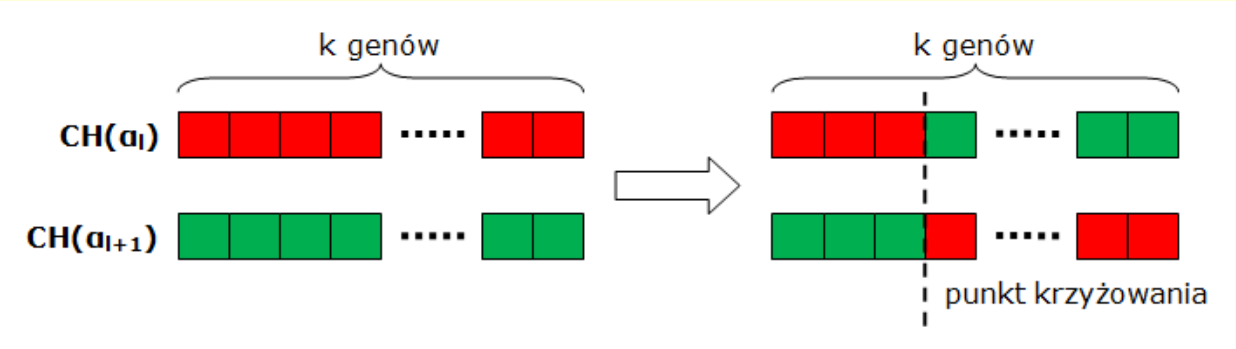
Definicja. *Operatorem selekcji probabilistycznej* $select$ nazywa funkcję

$$select(POP(\Delta,i)) = \left\{ \alpha \in POP(\Delta,i) \ : \ Pr(\text{ wystąpienie } \alpha) = \frac{adapt(\alpha)}{\sum\limits_{\beta \in POP(\Delta,i)} adapt(\beta)} \right\},$$

gdzie $select(POP(\Delta,i))$ jest i-elementowym multizbiorem.

Podejście zachłanne a algorytmy genetyczne – proces krzyżowania

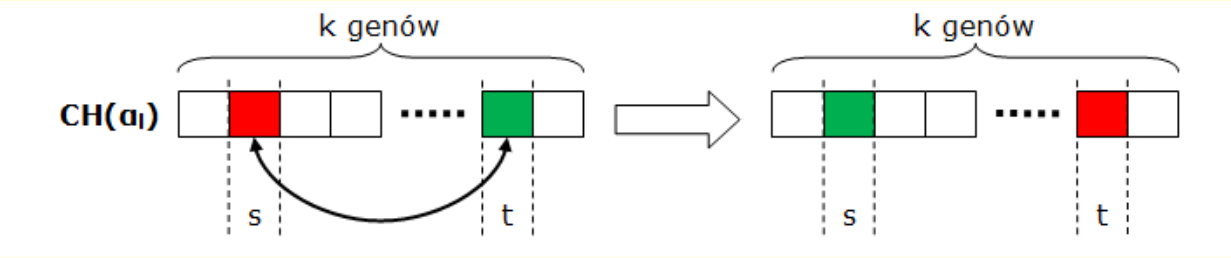
Definicja. *j-elementowym operatorem krzyżowania* $cross\left(POP\left(\Delta,i\right),j\right)$, gdzie $0 \leq j \leq i$ jest liczbą parzystą, nazywam funkcję, która dla j losowo wybranych osobników $\{\alpha_1, \alpha_2, \ldots, \alpha_j\}$ z populacji $POP\left(\Delta,i\right)$, krzyżuje chromosomy osobników α_l, α_{l+1} i $l = 1, 3, \ldots, j-1$ tak, że



gdzie punkt krzyżowania wybierany jest losowo z przedziału dyskretnego $[1, k]$.

Podejście zachłanne a algorytmy genetyczne – proces mutacji

Definicja. *Operatorem* mutacji $mutation\left(POP\left(\Delta,i\right),prob\right)$ nazywamy funkcję, która dla każdego osobnika α populacji $POP\left(\Delta,i\right)$ z zadanym prawdopodobieństwem $prob$ zamienia miejscami losowo wybrane geny osobnika α, $G\left(CH\left(\alpha\right),s\right)$ oraz $G\left(CH\left(\alpha\right),t\right)$, gdzie $1 \leq s,t \leq k$



Podejście zachłanne a algorytmy genetyczne

Uniwersalny schemat algorytmu genetycznego. Niech n będzie liczbą osobników w populacji, $time$ liczbą iteracji procesu reprodukcji (liczba analizowanych pokoleń), $cross\_nr$ liczbą osobników uczestniczących w procesie krzyżowania oraz $prob$ prawdopodobieństwem wystąpienia procesu mutacji

```
Individual GA(int n, int time, int cross_nr, real prob) {
   Individual Population[n];
   int i=0;

   GeneratePopulation(Population); // utworzenie populacji początkowej

   for (i=0;i<time;i++) {
      Population=Select(Population); // operator selekcji

      Population=Cross(Population,cross_nr); // operator krzyżowania

      Population=Mutation(Population,prob); // operator mutacji
   }

   return BestIndividual(Population); // wybieramy najlepszego osobnika
}
```

Podejście zachłanne a algorytmy genetyczne

Wniosek. Algorytm GA bazuje na podejściu „zachłannym", w każdym kroku algorytmu (iteracji pętli while) wybierany jest zbiór osobników o najlepszym dopasowaniu w odniesieniu do kryteriów poszukiwanego rozwiązania.

Wniosek. Podejście „zachłanne" w algorytmie GA jest istotnie inne od wcześniej rozważanych, tyczy się przekształcania zbioru rozwiązań (populacji), a nie konstrukcji pojedynczego rozwiązania. Stąd algorytm GA jest jest metodą „metazachłanną".

Wniosek. Algorytm GA jest algorytmem probabilistycznym, nie gwarantuje optymalności rozwiązania.

Fakt. Dla poprawnie dobranych argumentów wejściowych algorytm GA jest efektywnym rozwiązaniem problemu odkodowania obrazu.

Zadanie (****). Zaimplementuj algorytm GA dla problemu odkodowania obrazu z dostatecznie dużym k. Określ empirycznie wartości argumentów wejściowych, dla których efektywność metody jest maksymalna.