



Wykład 1 - Teoria grafów

Definicje

Graf $G = (V, E)$ składa się z **wierzchołków** V oraz **krawędzi** E .

Macierz przyległości (sąsiedztwa) oraz **macierz incydencji**:

Definicja

Niech $G = (V, E)$ będzie **grafem (prostym)**

o n wierzchołkach ($V = \{v_1, v_2, \dots, v_n\}$)

i m krawędziach ($E = \{e_1, e_2, \dots, e_m\}$).

- **Macierzą przyległości** grafu (prostego) G nazywamy macierz $n \times n$ $A(G) = [a_{i,j}]$, gdzie

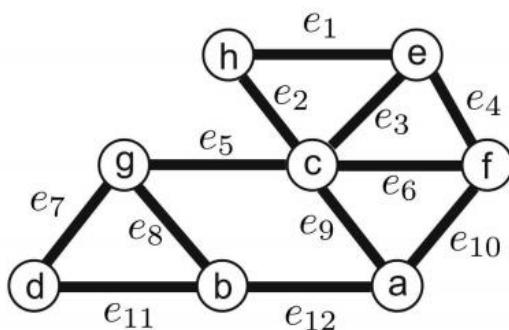
$$a_{i,j} = \begin{cases} 1, & \text{gdy } \{v_i, v_j\} \in E; \\ 0, & \text{gdy } \{v_i, v_j\} \notin E; \end{cases}$$

- **Macierzą incydencji** grafu (prostego) G nazywamy macierz $n \times m$ $M(G) = [m_{i,j}]$, gdzie

$$m_{i,j} = \begin{cases} 1, & \text{gdy } v_i \in e_j; \\ 0, & \text{gdy } v_i \notin e_j; \end{cases}$$

Przykład:

GRAF (GRAF PROSTY)



LISTA NASTĘPNIKÓW

- a: b, c, f
- b: a, d, g
- c: a, e, f, g, h
- d: b, g
- e: c, f, h
- f: a, c, e
- g: b, c, d
- h: c, e

MACIERZ PRZYLEGŁOŚCI

	a	b	c	d	e	f	g	h
a	0	1	1	0	0	1	0	0
b	1	0	0	1	0	0	1	0
c	1	0	0	0	1	1	1	1
d	0	1	0	0	0	0	1	0
e	0	0	1	0	0	1	0	1
f	1	0	1	0	1	0	0	0
g	0	1	1	1	0	0	0	0
h	0	0	1	0	1	0	0	0

MACIERZ INCYDENCJI

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
a	0	0	0	0	0	0	0	0	1	1	0	1
b	0	0	0	0	0	0	1	0	0	1	1	1
c	0	1	1	0	1	1	0	0	1	0	0	0
d	0	0	0	0	0	0	1	0	0	0	1	0
e	1	0	1	1	0	0	0	0	0	0	0	0
f	0	0	0	1	0	1	0	0	0	1	0	0
g	0	0	0	0	1	0	1	1	0	0	0	0
h	1	1	0	0	0	0	0	0	0	0	0	0

Zatem macierz przyległości (sąsiedztwa) dotyczy samych wierzchołków, macierz incydencji zaś wierzchołków i krawędzi (kolumny → krawędzie).



Uwaga! Dla multigrafów (Możliwe wiele krawędzi między tymi samymi wierzchołkami) wartości macierzy przyległości i incydencji wcale nie muszą być binarne!

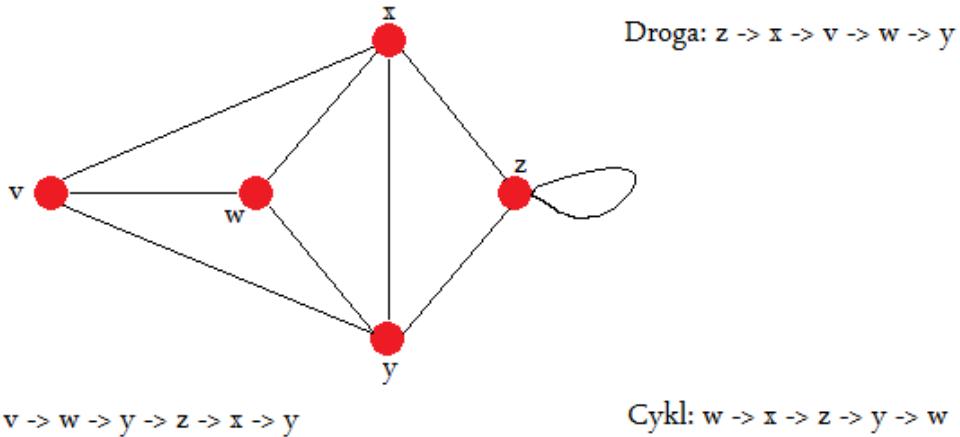
Stopień wierzchołka - liczba krawędzi incydentnych do wierzchołka (pętle liczone podwójnie). Innymi słowy - liczna sąsiadów wierzchołka.

Ścieżka - ścieżkę łączącą v_0 z v_n o długości n nazywa się ciąg wierzchołków (v_0, v_1, \dots, v_n) taki, że dla każdego $k \in \{0, 1, \dots, n-1\}$ istnieje krawędź z v_k do v_{k+1}

Droga – ścieżka, w której wierzchołki są różne.

Długość drogi/ścieżki – liczba krawędzi/wierzchołków, tworzących daną drogę/ścieżkę.

Cykł - droga zamknięta, czyli taka, której koniec (ostatni wierzchołek) jest identyczny z początkiem (pierwszym wierzchołkiem)



Graf G jest **spójny**, jeśli istnieje co najmniej jedna ścieżka między każdą parą wierzchołków w G (z każdego wierzchołka grafu da się dostać do dowolnego innego).

Graf G_1 jest **podgrafem** grafu G , jeśli wszystkie wierzchołki oraz wszystkie krawędzie z G_1 są w G oraz każda z krawędzi G_1 ma takie same wierzchołki końcowe jak i krawędzie w G . Intuicyjnie - mniejszy graf powstał z grafu G .

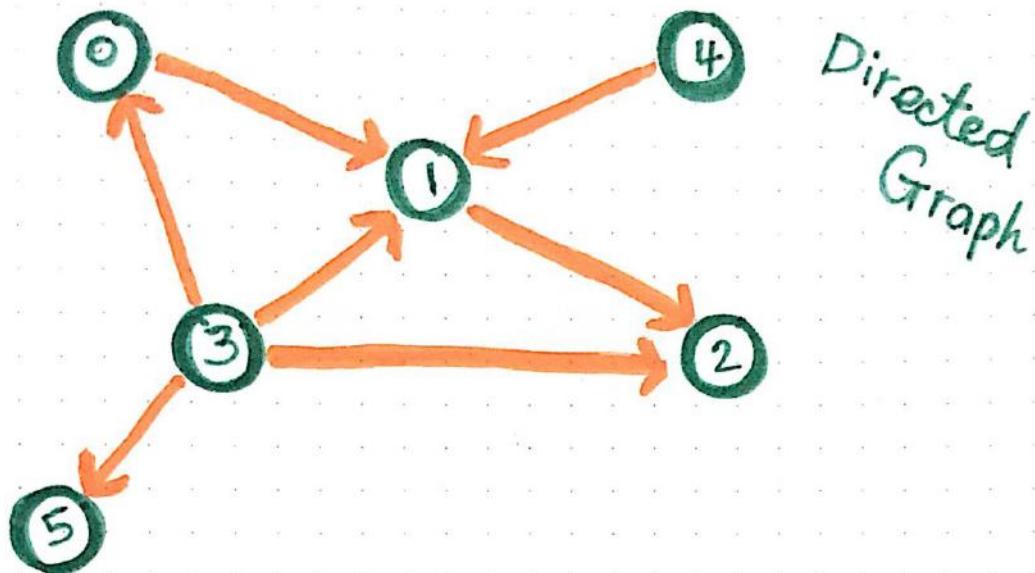
Graf ważony - każda krawędź ma przypisaną wagę.

Graf skierowany $G = (V, E)$ składa się z **wierzchołków V** oraz **krawędzi E**, które tym razem są **uporządkowane** (liczy się kolejność z którego wierzchołka wychodzi a do którego wchodzi).

Reprezentacje grafu

Reprezentacja grafu to sposób zapisu informatycznego grafu. Dwie najpopularniejsze:

- **Macierz sąsiedztwa** - to samo co macierz przyległości
- **Lista sąsiedztwa** - Tworzy się tablicę o rozmiarze równym liczbie wierzchołków, zawierającą wskaźniki na (początkowo puste) listy – kolejne elementy listy oznaczają będą kolejnych sąsiadów danego wierzchołka, do którego lista jest przyporządkowana. Reprezentacja użyteczna dla grafów rzadkich.



EDGE	0	0	1
1	1	2	
2	3	0	
3	3	1	
4	3	2	
5	3	5	
6	4	1	

LIST

A	0	1	2	3	4	5
D	0	0	1	0	0	0
J	1	0	0	1	0	0
A	2	0	0	0	0	0
C	3	1	1	1	0	1
T	4	0	1	0	0	0
R	5	0	0	0	0	0

MATRIX

0	1				
1	2				
2					
3	0	1	2	5	
4	1				
5					

ADJACENCY LIST

notice that this adjacency list contains a total of $|E|$ elements, one for each edge, since the edges are directed.

↑ notice that this matrix is not symmetrical!

Właściwości reprezentacji

Aa Name	Macierz sąsiedztwa	Lista sąsiedztwa
<u>Wymagania pamięciowe</u>	$O(V ^2)$	$O(E)$
<u>Dodanie nowej krawędzi</u>	czas stały	czas stały
<u>Sprawdzenie czy dana krawędź istnieje</u>	czas stały	$O(E)$
<u>Sprawdzenie stopnia danego wierzchołka</u>	$O(V)$	$O(E)$
<u>Usunięcie krawędzi</u>	czas stały	$O(E)$

Właściwości grafów

Cykl Hamiltona - cykl w grafie, który zawiera wszystkie jego wierzchołki (każdy wierzchołek tylko jeden raz). Grafy zawierające cykl Hamiltona to grafy hamiltonowskie.

!! Znalezienie cyklu Hamiltona o minimalnej sumie wag krawędzi jest równoważne rozwiązaniu zagadnienia komiwojażera.

Cykl Eulera - cykl w grafie, który zawiera wszystkie jego krawędzie (każda krawędź tylko jeden raz). Grafy zawierające cykl Hamiltona to grafy eulerowskie.

Acentryczność wierzchołka grafu - maksymalna z odległości tego wierzchołka do innych wierzchołków grafu. Acentryczność grafu nazywa się także jego ekscentrycznością.

Automorfizm grafu - wzajemne jednoznaczne przekształcenie zbioru wierzchołków grafu zachowujące sąsiedztwo.

Centrum grafu – wierzchołek grafu spójnego taki, że największa z odległości od centrum do innych wierzchołków grafu jest najmniejsza.

Drzewo spinające grafu – spójny, acykliczny podgraf danego grafu, zawierający wszystkie jego wierzchołki.

Gęstość grafu – stosunek liczby krawędzi do największej możliwej liczby krawędzi.

Klika – podzbiór wierzchołków danego grafu wraz z krawędziami je łączącymi, takich, że każde dwa wierzchołki tego podzbioru są sąsiadami.

Kolorowanie grafu – nadanie każdemu wierzchołkowi koloru tak, by żadne sąsiadujące ze sobą wierzchołki nie były pokolorowane tym samym kolorem.

Krawędź/wierzchołek krytyczny – krawędź/wierzchołek, po usunięciu której/którego ze zbioru pokrywającego zmniejsza się indeks pokrycia krawędziowego/wierzchołkowego.

Liczba chromatyczna – najmniejsza liczba kolorów potrzebna do prawidłowego pokolorowania grafu.

Most – krawędź, po usunięciu której wzrasta liczba spójnych składowych grafu.

Odległość – liczba krawędzi w najkrótszej ścieżce łączącej dane dwa wierzchołki.

Rozmiar grafu G – liczbę krawędzi grafu.

Rząd grafu G – liczba wierzchołków grafu.

Graf r-regularny – graf, w którym każdy wierzchołek grafu jest stopnia r.

Spójna składowa grafu G – możliwie największy spójny podgraf grafu G. Graf spójny ma jedną spójną składową.

Ściana – obszar zamknięty, wyznaczony przez krawędzie grafu (tzw. krawędzie tworzące ścianę).

Wierzchołek izolowany – wierzchołek o stopniu 0, czyli niebędący końcem żadnej krawędzi.

Wierzchołek rozcinający – wierzchołek, po usunięciu którego zwiększa się liczba spójnych składowych grafu.

Klasy grafów

Drzewo – spójny graf acykliczny.

Graf acykliczny – graf bez cyklu.

Graf dwudzielny – graf, którego wierzchołki mogą być podzielone na dwa zbiorów, tak by w obrębie jednego zbioru żaden wierzchołek nie był połączony z innym.

Graf k-dzielny – graf, którego zbiór wierzchołków można podzielić na k parami rozłącznych podzbiorów takich, że żadne dwa węzły, należące do tego samego zbioru, nie są połączone krawędzią.

Graf genusu G – graf, który można narysować bez przecięć (czyli w formie grafu płaskiego) na powierzchni genusu G, ale nie można narysować go bez przecięć na powierzchni genusu G-1. (???)

Graf k-spójny – graf, posiadający k spójnych składowych.

Graf pełny – graf, którego każdy wierzchołek jest połączony bezpośrednio krawędzią z każdym innym.

Graf krytyczny – graf, którego każdy wierzchołek/krawędź jest krytyczny/krytyczna.

Graf prosty - graf niezawierający pętli ani krawędzi wielokrotnych.

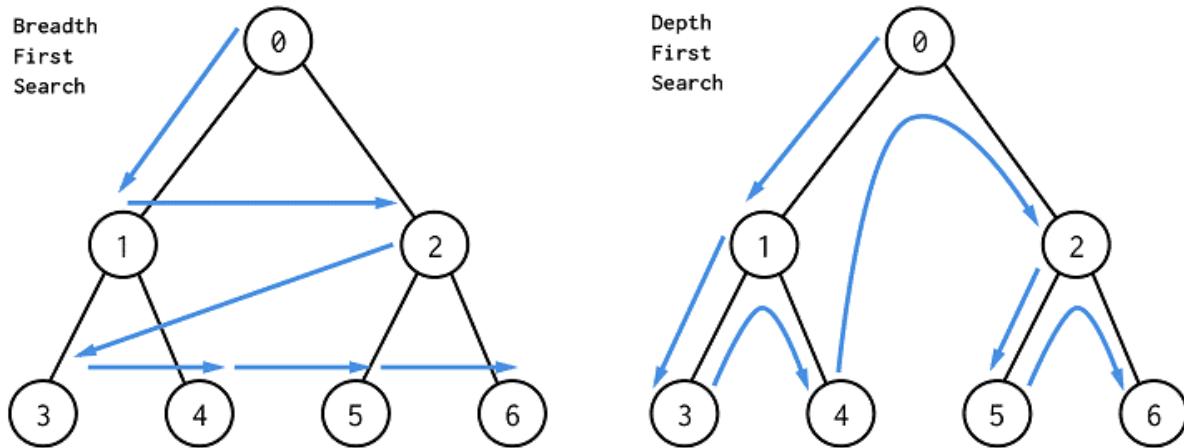
itp. reszta na Wikipedii XD

Przeszukiwanie grafu

Przeszukiwanie grafu to usystematyzowany sposób odwiedzenia wszystkich wierzchołków grafu np. w celu rozwiązania jakiegoś problemu.

Podstawowe metody przeszukiwania grafów:

- Przeszukiwanie wszerz (BFS)
- Przeszukiwanie w głąb (DFS)



Złożoność czasowa i pamięciowa: $O(|V| + |E|)$

Wykład 2

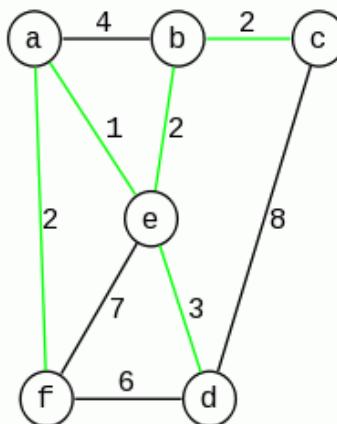
Drzewo – spójny graf acykliczny

Drzewo rozpinające

Drzewo, które zawiera wszystkie wierzchołki grafu G, zaś zbiór krawędzi drzewa jest podzbiorem zbioru krawędzi grafu. Konstrukcja drzewa rozpinającego polega na usuwaniu z grafu tych krawędzi, które należą do cykli.

Minimalne drzewo rozpinające (MSP) – drzewo rozpinające danego grafu o najmniejszej z możliwych wag.

Przykładowe MSP:



Algorytmy znajdujące minimalne drzewo rozpinające:

1. **Boruvki (Sollina)**
2. **Prima (Dijkstry – Prima)** – $O(E^* \log V)$, przy zastosowaniu kopca Fibonacciego
 $O(E + V \log V)$
3. **Kruskala**

```
G: Graf
V: zbiór wierzchołków G
a: funkcja wag krawędzi a[u,v]
s: wierzchołek startowy
Q: zbiór wierzchołków nienależących do MST
A: zbiór krawędzi MST
alfa[u]: poprzednik wierzchołka u w MST
beta[u]: waga krawędzi łączącej u z MST (z wierzchołkiem alfa[u])
N[u]: lista sąsiedztwa wierzchołka u
```

u, u^* : bieżący i ostatnio wybrany wierzchołek
 suma: sumaryczna waga krawędzi MST

```
DPA(G, a, s)

01  suma ← 0
02  A ← ∅
03  dla każdego  $u \in V$ 
04      alfa[u] ← 0
05      beta[u] ← ∞
06  Q ← V
07  beta[s] ← 0
08  Q ← Q-{s}
09  u* ← s
10  dopóki Q ≠ ∅
11      dla każdego ( $u \in Q$  i  $u \in N[u^*]$ )
12          jeśli  $a[u, u^*] < \beta[u]$  to alfa[u] ←  $u^*$ ; beta[u] ←  $a[u, u^*]$ 
13      dla każdego  $u \in Q$ 
14           $u^* \leftarrow \arg \min(\beta[u])$ 
15      Q ← Q-{ $u^*$ }
16      A ← A + {alfa[ $u^*$ ],  $u^*$ }
17      suma ← suma + a[alfa[ $u^*$ ],  $u^*$ ]
18  zwróć (A, suma)
```

Liczba drzew rozpinających grafu

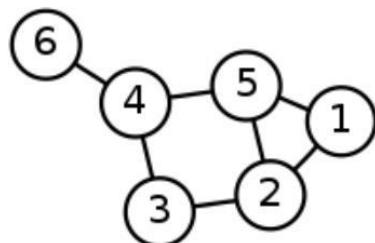
Twierdzenie Kirchhoffa

Tworzymy macierz A, taką, że:

$$a_{ij} = \begin{cases} \deg(v_i) & \text{dla } i = j \\ -1 & \text{dla } i \neq j \text{ oraz } v_i \text{ sasiaduje z } v_j \\ 0 & \text{w pozostałych przypadkach} \end{cases}$$

Przykładowo:

Graf:



Macierz:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Liczymy dopełnienie algebraiczne dowolnego elementu (np. tu 11) i otrzymujemy wynik.

$$A_{11} = (-1)^{1+1} \cdot \begin{vmatrix} 3 & -1 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{vmatrix} = 11$$

Poszukiwanie najkrótszej ścieżki między dwoma wierzchołkami

Algorytmy:

1. Dijkstry $O(V^2)$ – brak wag ujemnych
2. A* - $O(\log(h^*(x)))$ gdzie h^* jest optymalną heurystyką, czyli zawsze podaje dokładny, rzeczywisty koszt ścieżki z x do węzła końcowego. Innymi słowy, błąd funkcji h nie powinien rosnąć szybciej niż logarytm „dokładnej heurystyki” h^* . (tu * to nie jest mnożenie!)
3. Bellmana – Forda $O(V^2E)$
4. Floyda – Warshalla $O(V^3)$ – pomiędzy wszystkimi parami wierzchołków
5. Johnsona $O(V^2 \log V + V^2 E)$ – pomiędzy wszystkimi parami wierzchołków

Algorytm Dijksty

Algorytm ten znajduje najkrótszą ścieżkę od jednego do pozostałych wierzchołków

```
Dijkstra(G,w,s):
    dla każdego wierzchołka v w V[G] wykonaj
        d[v] := nieskończoność
        poprzednik[v] := niezdefiniowane
    d[s] := 0
    Q := V
    dopóki Q niepuste wykonaj
        u := Zdejmij_Min(Q)
        dla każdego wierzchołka v – sąsiada u wykonaj
            jeżeli d[v] > d[u] + w(u, v) to
                d[v] := d[u] + w(u, v)
                poprzednik[v] := u
    Wyświetl("Droga wynosi: " + d[v])
```

Złożoność alg. Dijkstry:

- dla zwykłej tablicy $O(V^2)$
- dla kopca $O(E \log V)$
- dla kopca Fibonacciego $O(V^* \log V + E)$.

Algorytm Floyda – Warsha

G – graf

w – tablica wagowa

```
Floyd-Warshall( $G, w$ )  
  
dla każdego wierzchołka  $v_1$  w  $V[G]$  wykonaj  
    dla każdego wierzchołka  $v_2$  w  $V[G]$  wykonaj  
         $d[v_1][v_2] = \text{nie skończone}$   
         $\text{poprzednik}[v_1][v_2] = \text{niezdefiniowane}$   
         $d[v_1][v_1] = 0$   
    dla każdej krawędzi  $(v_1, v_2)$  w  $E[G]$   
         $d[v_1][v_2] = w(v_1, v_2)$   
         $\text{poprzednik}[v_1][v_2] = v_1$   
    dla każdego wierzchołka  $u$  w  $V[G]$  wykonaj  
        dla każdego wierzchołka  $v_1$  w  $V[G]$  wykonaj  
            dla każdego wierzchołka  $v_2$  w  $V[G]$  wykonaj  
                jeśli  $d[v_1][v_2] > d[v_1][u] + d[u][v_2]$  to  
                     $d[v_1][v_2] = d[v_1][u] + d[u][v_2]$   
                     $\text{poprzednik}[v_1][v_2] = \text{poprzednik}[u][v_2]$ 
```

Algorytmy poszukiwania najkrótszej ścieżki w grafie

Spis treści

1. Algorytm Floyda-Warshalla	2
a) Złożoność: obliczeniowa $O(n^3)$, pamięciowa $O(n^2)$	2
b) Ograniczenia.....	2
c) Rezultat: wyznaczenie najkrótszych ścieżek pomiędzy wszystkimi wierzchołkami	2
d) Zasada działania	2
e) Pseudokod	3
2. Algorytm A*.....	4
a) Złożoność: (niepodana na wykładzie) obliczeniowa $O(E)$, pamięciowa $O(V)$	4
b) Ograniczenia:.....	4
c) Rezultat: znalezienie najkrótszej ścieżki pomiędzy dwoma wierzchołkami.....	4
d) Zasada działania	4
e) Pseudokod	5
3. Alg. Bellmana-Forda.....	6
f) Złożoność: (niepodana na wykładzie) czasowa $O(V \cdot E)$, pamięciowa $O(V)$	6
g) Ograniczenia:.....	6
h) Rezultat: najkrótsze ścieżki od wierzchołka startowego do wszystkich pozostałych wierzchołków (można nim wykryć cykle ujemne).....	6
i) Zasada działania:	6
j) Pseudokod	7
4. Alg. Johnsona.....	8
a) Złożoność: (nie podana na wykładzie) czasowa $O(V^2 \log V + VE)$	8
b) Ograniczenia:.....	8
c) Rezultat: wyznaczenie najkrótszej drogi pomiędzy wszystkimi wierzchołkami w grafie.	8
d) Zasada działania:	8
e) Pseudokod	9
5. Przydatne strony:.....	10

1. Algorytm Floyda-Warshalla

a) Złożoność: obliczeniowa $O(n^3)$, pamięciowa $O(n^2)$

b) Ograniczenia

- Dopuszczalne krawędzie ujemne
- Niedozwolone cykle ujemne
- Na wykładzie omawiany dla grafu skierowanego

c) Rezultat: wyznaczenie najkrótszych ścieżek pomiędzy wszystkimi wierzchołkami

d) Zasada działania

(przekopiowany z Wikipedii, ale chyba jest wystarczająco jasno wytłumaczone)

Algorytm Floyda-Warshalla korzysta z tego, że jeśli najkrótsza ścieżka pomiędzy wierzchołkami v_1 i v_2 prowadzi przez wierzchołek u , to jest ona połączeniem najkrótszych ścieżek pomiędzy wierzchołkami v_1 i u oraz u i v_2 . Na początku działania algorytmu inicjowana jest tablica długości najkrótszych ścieżek, tak że dla każdej pary wierzchołków (v_1, v_2) ich odległość wynosi:

$$d[v_1, v_2] = \begin{cases} 0, & \text{gdy } v_1 = v_2 \\ w(v_1, v_2), & \text{gdy } (v_1, v_2) \in E \\ +\infty, & \text{gdy } (v_1, v_2) \notin E \end{cases}$$

Algorytm jest dynamiczny i w kolejnych krokach włącza do swoich obliczeń ścieżki przechodzące przez kolejne wierzchołki. Tak więc w k -tym kroku algorytm zajmie się sprawdzaniem dla każdej pary wierzchołków, czy nie da się skrócić (lub utworzyć) ścieżki pomiędzy nimi przechodzącej przez wierzchołek numer k (kolejność wierzchołków jest obojętna, ważne tylko, żeby nie zmieniała się w trakcie działania programu). Po wykonaniu $|V|$ takich kroków długości najkrótszych ścieżek są już wyliczone.

e) Pseudokod

```
Floyd-Warshall( $G, w$ )  
  
dla każdego wierzchołka  $v_1$  w  $V[G]$  wykonaj  
    dla każdego wierzchołka  $v_2$  w  $V[G]$  wykonaj  
         $d[v_1][v_2] = \text{nie skończone}$   
         $\text{poprzednik}[v_1][v_2] = \text{niezdefiniowane}$   
         $d[v_1][v_1] = 0$   
dla każdej krawędzi  $(v_1, v_2)$  w  $E[G]$   
     $d[v_1][v_2] = w(v_1, v_2)$   
     $\text{poprzednik}[v_1][v_2] = v_1$   
dla każdego wierzchołka  $u$  w  $V[G]$  wykonaj  
    dla każdego wierzchołka  $v_1$  w  $V[G]$  wykonaj  
        dla każdego wierzchołka  $v_2$  w  $V[G]$  wykonaj  
            jeśli  $d[v_1][v_2] > d[v_1][u] + d[u][v_2]$  to  
                 $d[v_1][v_2] = d[v_1][u] + d[u][v_2]$   
                 $\text{poprzednik}[v_1][v_2] = \text{poprzednik}[u][v_2]$ 
```

2. Algorytm A*

a) **Złożoność: (niepodana na wykładzie) obliczeniowa $O(|E|)$, pamięciowa $O(|V|)$**

Worst-case performance $O(|E|) = O(b^d)$

Worst-case space complexity $O(|V|) = O(b^d)$

b) **Ograniczenia:**

(prawdopodobnie – tzn, tak wydaje się logicznie, ale nie znalazłem tego napisanego explicite)

- Dopuszczalne krawędzie ujemne
- Niedozwolone cykle ujemne

(Na pewno)

- Właściwa heuretyka, czyli taka, która nigdy nie zawyży odległości pomiędzy dwiema wierzchołkami

c) **Rezultat: znalezienie najkrótszej ścieżki pomiędzy dwoma wierzchołkami**

d) **Zasada działania**

Algorytm minimalizuje funkcję kosztu $f(x) = g(x) + h(x)$ – gdzie $g(x)$ to faktycznie przebyta odległość od wierzchołka startowego do x , a $h(x)$ to szacowana optymistyczna (czyli nie większa od rzeczywistej) odległość do wierzchołka końcowego. W każdym kroku wybiera kolejny wierzchołek x minimalizujący wspomnianą funkcję $f(x)$

e) Pseudokod

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how short a path from start to finish can be if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(1) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

3. Alg. Bellmana-Forda

f) Złożoność: (niepodana na wykładzie) czasowa $O(|V| \cdot |E|)$, pamięciowa $O(|V|)$

g) Ograniczenia:

- Dopuszczalne wagi ujemne
- Niedopuszczalne cykle ujemne

h) Rezultat: najkrótsze ścieżki od wierzchołka startowego do wszystkich pozostałych wierzchołków (można nim wykryć cykle ujemne)

i) Zasada działania:

W trakcie wykonywania algorytmu dla każdego wierzchołka zostają wyznaczone dwie wartości: koszt dotarcia do tego wierzchołka (oznaczmy go jako d_i) oraz poprzedni wierzchołek na ścieżce (oznaczmy go jako p_i). Na początku działania algorytmu dla wierzchołka źródłowego koszt dotarcia wynosi 0 (już tam jesteśmy), a dla każdego innego wierzchołka nieskończoność (w ogóle nie wiemy, jak się tam dostać).

Następnie dla każdej krawędzi (oznaczmy, że aktualnie analizowana krawędź ma wagę k i prowadzi z wierzchołka u do wierzchołka v) wykonujemy następującą czynność:

Jeżeli $d_v > d_u + k$, to ustawiamy wartość d_v na $d_u + k$, a wartość p_v na u .

Całość (przejście wszystkich krawędzi) należy powtórzyć $n-1$ razy, gdzie n jest liczbą wierzchołków. W każdej iteracji należy przejrzeć wszystkie krawędzie w tej samej kolejności. Jeśli w którejś iteracji nie nastąpią żadne zmiany, wykonywanie algorytmu można przerwać wcześniej.

Wykrywanie ujemnych cykli

Po zakończeniu działania algorytmu, dla każdej krawędzi powinna być spełniona nierówność: $d_v \leq d_u + k$ (k to waga krawędzi, krawędź prowadzi z wierzchołka u do v). Mówiąc inaczej, kolejna iteracja algorytmu nie powinna spowodować jakichkolwiek zmian. Jeśli ten warunek nie jest spełniony, to w grafie występuje ujemny cykl osiągalny z wierzchołka źródłowego.

Aby sprawdzić występowanie ujemnych pętli w całym grafie, należy dodać do grafu nowy wierzchołek, połączyć go krawędziami o zerowej wadze ze wszystkimi innymi wierzchołkami, a następnie wykonać algorytm traktując ten nowy wierzchołek jako wierzchołek źródłowy. Można również po prostu wykonać algorytm n razy (za każdym razem dla innego wierzchołka źródłowego), jednak zwiększyłoby to złożoność obliczeniową.

j) Pseudokod

Z przykładu:

```
Bellman-Ford( $G, w, s$ ) :  
  
    dla każdego wierzchołka  $v \in V$  wykonaj  
         $d[v] = \infty$   
         $p[v] = -1$   
         $d[s] = 0$   
    dla  $i$  od 1 do  $|V|-1$  wykonaj  
        dla każdej krawędzi  $(u, v) \in E$  wykonaj  
            jeśli  $d[v] > d[u] + w(u, v)$  to  
                 $d[v] = d[u] + w(u, v)$   
                 $p[v] = u$ 
```

Z angielskiej Wikipedii:

```
function BellmanFord(list vertices, list edges, vertex source) is  
  
    // This implementation takes in a graph, represented as  
    // lists of vertices (represented as integers [0..n-1])  
    // and edges,  
    // and fills two arrays (distance and predecessor)  
    // holding  
    // the shortest path from the source to each vertex  
  
    distance := list of size n  
    predecessor := list of size n  
  
    // Step 1: initialize graph  
    for each vertex v in vertices do  
        distance[v] := inf           // Initialize the  
        distance to all vertices to infinity  
        predecessor[v] := null       // And having a null  
        predecessor  
  
        distance[source] := 0          // The distance from  
        the source to itself is, of course, zero  
  
    // Step 2: relax edges repeatedly  
    repeat |V|-1 times:  
        for each edge (u, v) with weight w in edges do  
            if distance[u] + w < distance[v] then  
                distance[v] := distance[u] + w  
                predecessor[v] := u  
  
    // Step 3: check for negative-weight cycles  
    for each edge (u, v) with weight w in edges do  
        if distance[u] + w < distance[v] then  
            error "Graph contains a negative-weight cycle"  
  
    return distance, predecessor
```

4. Alg. Johnsona

a) **Złożoność: (nie podana na wykładzie) czasowa**

$$O(|V^2| \log(|V|) + |V||E|)$$

b) **Ograniczenia:**

- Dopuszczalne wagi ujemne
- Niedopuszczalne cykle ujemne

c) **Rezultat: wyznaczenie najkrótszej drogi pomiędzy wszystkimi wierzchołkami w grafie.**

d) **Zasada działania:**

Na początku wykonujemy algorytm Bellmana-Forda w wersji z dodatkowym wierzchołkiem. W ten sposób weryfikujemy, czy graf nie zawiera ujemnych cykli. Oznaczmy odległości znalezione przez ten algorytm jako h_i (i jest wybranym wierzchołkiem grafu).

Następnie trzeba zmodyfikować wagi krawędzi tak, aby pozbyć się wartości ujemnych, a jednocześnie nie zmienić optymalnych tras pomiędzy wierzchołkami. W tym celu możemy skorzystać ze wzoru $k'i,j = k_{i,j} + h_i - h_j$, gdzie $k_{i,j}$ jest wagą krawędzi prowadzącej z wierzchołka i do wierzchołka j , a wartości h_i i h_j są rezultatami wykonania algorytmu Bellmana-Forda.

Teraz nie mamy już krawędzi o ujemnych wagach, możemy zatem wykonać algorytm Dijkstry. Algorytm ten wykonujemy n razy, za każdym razem biorąc inny wierzchołek jako źródłowy. W ten sposób znajdujemy najkrótsze ścieżki pomiędzy każdą parą wierzchołków.

Na końcu musimy odtworzyć oryginalne odległości. W tym celu korzystamy ze wzoru: $d_{i,j} = d'i,j - h_i + h_j$, gdzie $d'i,j$ jest długością ścieżki z wierzchołka i do wierzchołka j wyznaczoną przez algorytm Dijkstry.

e) Pseudokod

Z Internetu

JOHNSON(G, w)

- 1 compute G' , where $G'.V = G.V \cup \{s\}$,
 $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
 $w(s, v) = 0$ for all $v \in G.V$
- 2 if BELLMAN-FORD(G', w, s) == FALSE
3 print “the input graph contains a negative-weight cycle”
- 4 else for each vertex $v \in G'.V$
 - 5 set $h(v)$ to the value of $\delta(s, v)$
computed by the Bellman-Ford algorithm
 - 6 for each edge $(u, v) \in G'.E$
 - 7 $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
 - 8 let $D = (d_{uv})$ be a new $n \times n$ matrix
 - 9 for each vertex $u \in G.V$
 - 10 run DIJKSTRA(G, \hat{w}, u) to compute $\hat{\delta}(u, v)$ for all $v \in G.V$
 - 11 for each vertex $v \in G.V$
 - 12 $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
- 13 return D

Opis słowny z wykładu

Algorytm Johnsona

Dopuszczalne w grafie G wagi ujemne k_{ij} (brak ujemnych cykli)

1. Realizacja algorytmu Bellmana-Forda w grafie z dodatkowym wierzchołkiem.
 - sprawdzenie, czy graf nie zawiera ujemnych cykli.
 - h_i – wyznaczona odległość dla i -tego wierzchołka przez BF (0).
2. Modyfikacja wag krawędzi
 - pozbycie się wartości ujemnych, bez zmiany optymalnych tras pomiędzy wierzchołkami
 - Obliczamy dla każdej krawędzi (i,j) : $k'_{ij} = k_{ij} + h_i - h_j$
 - Usuwamy dodatkowy wierzchołek
3. n razy wykonać algorytm Dijkstry – dla wszystkich wierzchołków startowych
 - znajdujemy najkrótsze ścieżki pomiędzy każdą parą wierzchołków.
4. Wyznaczenie odległości dla grafu G .
 - $d_{ij} = d'_{ij} - h_i + h_j$
 - gdzie d'_{ij} jest długością ścieżki z wierzchołka i do j wyznaczoną przez algorytm Dijkstry.

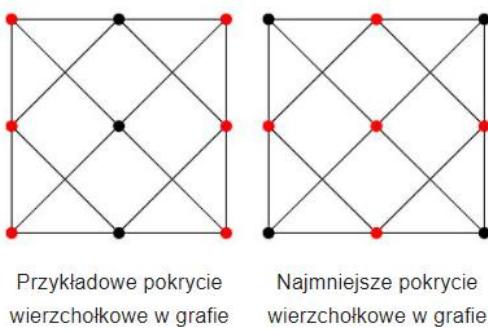
5. Przydatne strony:

- <http://algorytmyENCY.pl/>
- <https://www.geeksforgeeks.org/johnsons-algorithm-for-all-pairs-shortest-paths-implementation/>

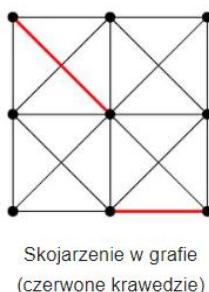
Wykład 4

1. Problemy grafowe

- Zagadnienie Komiwojażera (ang. Traveling Salesman) - Problem wędrownego sprzedawcy - należy znaleźć najkrótszą (najtańszą/najszybszą) ścieżkę łączącą wszystkie wierzchołki grafu. Ścieżka musi zaczynać się i kończyć w określonym punkcie. (Należy znaleźć minimalny cykl Hamiltona w pełnym grafie ważonym).
- Najdłuższa ścieżka (ang. Longest Path) - ścieżka (droga łącząca 2 wierzchołki, w której krawędzie się nie powtarzają), której koszt przejścia jest największy. Wykorzystuje się to w programowaniu sieciowym (CPM, PERT), w zarządzaniu projektem (znalezienie ścieżki krytycznej - ciąg czynności których opóźnienie powoduje wydłużenie czasu realizacji całości projektu).
- Pokrycie wierzchołkowe (ang. Vertex Cover) - taki podzbiór wierzchołków grafu, w którym każda krawędź jest incydentna (krawędź jest incydentna z wierzchołkiem, jeśli ma w tym wierzchołku początek lub koniec) do jakiegoś wierzchołka z tego podzbioru.



- Zbiór dominujący (ang. Dominating Set) - podzbiór wierzchołków grafu taki, że każdy wierzchołek, który nie należy do tego zbioru ma w nim co najmniej jednego sąsiada (są połączone krawędzią).
- Skojarzenie (ang. Matching) - podzbiór krawędzi grafu taki, że każdy wierzchołek jest końcem co najwyżej jednej krawędzi. Wierzchołki, będące końcami krawędzi są należących do M są M-nasycone, pozostałe są M-nienasycone. Skojarzenie doskonałe to takie, gdzie każdy wierzchołek jest M-nasycony



Skojarzenie w grafie
(czerwone krawędzie)

- Podział grafu (ang. Graph Partitioning) - redukcja grafu do grafu mniejszego poprzez podział jego węzłów w wzajemnie wykluczające się grupy.

- Kolorowanie grafu (ang. Graph Coloring) - przypisanie określonym elementom składowym grafu (najczęściej wierzchołkom, rzadziej krawędziom) wybranych kolorów według ściśle określonych reguł.
- Izomorfizm grafu (ang. Graph Isomorphism) - dwa grafy są izomorficzne, jeśli istnieje takie przeetykietowanie wierzchołków jednego z nich, że jeśli dwa wierzchołki w jednym są połączone krawędzią to w drugim też istnieje takie połączenie. (Grafy są identyczne pod względem matematycznym (np. ilość wierzchołków, krawędzi itp.), a jedynie różnią się wizualnie (inaczej rozmieszczone wierzchołki)).
- Zagadnienie przydziału (ang. Assignment Problem) - problem polegający na odpowiednim rozplanowaniu posiadanych zasobów do prac (np. przydział robotników do pracy w taki sposób, żeby skończyć ją jak najszybciej/najtaniej).
- Minimalne drzewo rozpinające grafu (MST) - drzewo rozpinające (podzbiór krawędzi grafu, po którym z każdego wierzchołka możemy przejść do każdego innego ale tylko w jeden sposób) o najmniejszej możliwej sumie wag.
- Najkrótsza ścieżka (SP - ang. Shortest Path) - ścieżka (droga łącząca 2 wierzchołki, w której krawędzie się nie powtarzają), której koszt przejścia jest najmniejszy.

2. Model matematyczny

Model matematyczny procesu (np. zjawiska fizycznego, systemu produkcji) składa się z:

- zbioru zmiennych decyzyjnych oraz parametrów opisujących problem
- funkcji celu będącej matematycznym zapisem kryterium optymalizacyjnego
- zbioru ograniczeń

Modele matematyczne podzielić można na:

Ze względu na wyniki:

- deterministyczne - wyniki są określone jednoznacznie
- niedeterministyczne - wyniki zależą od wielkości losowych i mogą być przewidziane zgodnie z zasadami probabilistyki
- model wartości oczekiwanych - wielkościom losowym zostały nadane ich wartości oczekiwane

Ze względu na charakter zbioru zmiennych decyzyjnych:

- model optymalizacji dyskretnej (zbior zmiennych decyzyjnych jest skończonym zbiorem wartości dyskretnych)
- model optymalizacji ciągłej (bez ograniczeń zakresu zmiennych)

Ze względu na liczbę funkcji celów:

- model optymalizacji skalarnej (tylko jedna funkcja celu)
- model optymalizacji wielokryterialnej (wektorowej) (kilka funkcji celu)

Ze względu na rodzaj funkcji celu oraz ograniczeń:

- model liniowy (funkcja celu i wszystkie ograniczenia są liniowe)
- model nieliniowy (funkcja celu, lub chociaż jedno z ograniczeń ma charakter nieliniowy)
- dyskretny - zmienne decyzyjne są dyskretne
- permutacyjny - zmienne decyzyjne są permutacją pewnego zbioru

Budowa modelu matematycznego obejmuje:

- określenie zmiennych decyzyjnych
- określenie funkcji celu
- określenie dopuszczalnego obszaru rozwiązań

3. Zagadnienie komiwojażera

Zagadnienie komiwojażera - Problem polegający na znalezieniu minimalnego cyklu Hamiltona w grafie ważonym. Nazwa pochodzi od wędrownego sprzedawcy (komiwojażera), który ma odwiedzić dokładnie jeden raz każde z "n" miast (gdzie znana jest odległość pomiędzy miastami) i wrócić do miasta początkowego. Dane są zatem zbiór wierzchołków $N = \{1, \dots, n\}$ i macierz odległości ($n \times n$) i należy znaleźć permutację zbioru N , która minimalizuje funkcję celu.

Cykł Hamiltona to taki cykl w grafie, w którym każdy wierzchołek grafu występuje jeden raz. Znalezienie cyklu Hamiltona o minimalnej sumie wag krawędzi jest równoważne rozwiązańu zagadnienia komiwojażera. Graf zawierający cykl Hamiltona nazywamy hamiltonowskim.

Cykł Eulera to taka zamknięta droga w grafie, która przechodzi przez każdą jego krawędź dokładnie jeden raz. Graf, który zawiera cykl Eulera nazywamy eulerowskim.

Symetryczne zagadnienie komiwojażera (STSP) polega na tym, że odległość od miasta A do B jest równa odległości B do A.

Asymetryczne zagadnienie komiwojażera (ATSP) polega na tym, że odległości z miasta A do B i z B do A mogą się różnić.

Otwarte zagadnienie komiwojażera (OSTP) nie tworzy cyklu, wierzchołek początkowy i końcowy nie jest zadany.

Problem TSP można rozwiązać korzystając z kilku algorytmów:

Algorytmy dokładne - tylko dla instancji o małym rozmiarze:

- Przegląd zupełny - dla TSP ma złożoność obliczeniową ($n!$) co nie pozwala na rozwiązywania instancji o rozmiarze $n > 20$ wierzchołków.
- Programowanie dynamiczne
- Metoda podziału i ograniczeń (ang. Little'a)
- Alg. Estmana - relaksacja problemu TSP do AP

Algorytmy przybliżone - rozwiązania są suboptimalne w akceptowalnym czasie:

- Alg. najbliższego sąsiada

- Alg. GTSP, FARIN, NERIN
- Alg. przeszukiwania losowego
- Alg. Ewolucyjne

Algorytm najbliższego sąsiada (NN)

TSP – algorytm najbliższego sąsiada (NN)

- Algorytm zachłanny
- Złożoność $o(n^2)$ – rozwiązanie nieoptymalny.
- Dolne ograniczenie dla instancji.

Etapy algorytmu:

1. Start z dowolnego wierzchołka (wierzchołek aktualny), który markujemy jako odwiedzony (wstawienie do rozwiązania).
2. Znajdź krawędź o najmniejszej wadze łączącą wierzchołek aktualny z nieodwiedzonymi wierzchołkami - v.
3. Przejdź do v (wierzchołek aktualny).
4. Oznacz v jako odwiedzony (wstawienie do rozwiązania).
5. Jeżeli wszystkie wierzchołki V są odwiedzane: STOP – zwróć sekwencję odwiedzonych wierzchołków.
6. Idź do kroku 2 .

Algorytm zachłanny (G_TSP)

Algorytm zachłanny – G_TSP

G_TSP – ang. *greed TSP*

Krok 1: Uporządkuj łuki (krawędzie) wg wag w ciąg niemalejący:

$$a_{e_1} \leq a_{e_2} \leq \dots \leq a_{e_n}$$

gdzie : $e_j \in E$

m - liczba łuków, tzn. $m = |E|$

Krok 2: Dla $j=1$ do m wykonaj:

dołącz e_j do rozwiązania, jeżeli nie powoduje to powstania podcyklu (podkonturu)

- Złożoność obliczeniowa *kroku 1* – sortowanie: $O(m * \log m)$
- Złożoność obliczeniowa *kroku 2* – $O(m)$
- Dla zagadnienia asymetrycznego A_TSP: $m \leq (n-1)*n$
- Dla zagadnienia symetrycznego S_TSP: $m \leq 1/2 * (n-1)*n$

Algorytm FARIN

TSP – FARIN

- *Farthest Insertion Heuristik*
 - Algorytm zachłanny – wstawienia najdalszego wierzchołka
 - Złożoność $o(n^2)$ – rozwiązanie nieoptymalny.
 - Dolne ograniczenie dla instancji.
1. Algorytm zaczyna się od rozwiązania (drogi komiwojażera), które składa się z jednego, losowo wybranego węzła.
 2. Wybierz "najdroższy" wierzchołek v (wierzchołek nieoznaczony najdalszy od aktualnej trasy)
 3. Wstaw v do rozwiązania w "najtańszym" miejscu sekwencji
 4. Jeżeli wszystkie wierzchołki V są odwiedzane: STOP – zwróć sekwencję odwiedzonych wierzchołków.
 5. Idź do kroku 2.

Algorytm NEARIN

TSP – NEARIN

- *Nearest-Insertion-Heuristik*
 - Algorytm zachłanny – wstawienia najbliższego wierzchołka
 - Złożoność $O(n^2)$ – rozwiązanie nieoptymalny.
 - Dolne ograniczenie dla instancji.
1. Algorytm zaczyna się od rozwiązania (drogi komiwojażera), które składa się z jednego, losowo wybranego węzła.
 2. Wybierz najbliższy wierzchołek v (wierzchołek nieoznaczony najbliższy do wierzchołków rozwiązania)
 3. Wstaw v do rozwiązania w "najtańszym" miejscu sekwencji
 4. Jeżeli wszystkie wierzchołki V są odwiedzane: STOP – zwróć sekwencję odwiedzonych wierzchołków.
 5. Idź do kroku 2.

Heurystyki dla TSP

Algorytm (heurystyka)	Typ	Złożoność obliczeniowa
Nearest-/ Farthest- Neighbor (NN, FN)	Konstrukcyjny	$O(n^2)$
Farthest-Insertion (FARIN)	Konstrukcyjny	$O(n^2)$
Nearest-Insertion (NEARIN)	Konstrukcyjny	$O(n^2)$
Minimum Spanning Tree	Konstrukcyjny	$O(n^2 \log(n))$
Heurystyka Christofidesa	Konstrukcyjny	$O(n^3)$
K-opt	Poprawy	$O(k!)$ – każdy krok
Suma n najkrótszych krawędzi (LB)	Heurystyki podwójne	$O(n^2 \log(n))$
Długość minimalnego drzewa rozpinającego (MST + 2-Matching)	Heurystyki podwójne	$O(n^2 \log(n))$

Problem VRP

Problem marszrutyzacji (Vehicle Routing Problem - VRP) jest rozszerzeniem problemu TSP i polega na wyznaczeniu optymalnych tras przewozowych dla pewnej ściśle określonej liczby środków transportu. Istnieje możliwość rozszerzenia problemu VRP o dodatkowe warunki:

Rozszerzenia problemu VRP

- problemy uwzględniające niesymetryczność kosztów przewozu pomiędzy wierzchołkami,
- problemy uwzględniające niehomogeniczność taboru,
- problemy uwzględniające przejazdy drobnicowe (Less Than Truckload),
- problemy uwzględniające ograniczenie maksymalnej długości trasy,
- problemy umożliwiające ustalenie baz (jednej lub kilku), w których pojazdy zaczynają i kończą podróż (Multiple Depot VRP),
- problemy umożliwiające dodanie baz pomocniczych (VRP with Satellite Facilities),
- problemy umożliwiające ustalenie częstotliwości odbioru/dostawy ładunku,
- problemy umożliwiające uwzględnienie okien czasowych (VRP with Time Windows) odbioru/wysłania towaru,

- problemy wiążące problem marszrutyzacji z problemem kontroli zapasów u klientów,
- problemy uwzględniające możliwość obsługi jednego klienta przez kilka pojazdów (Split Delivery VRP),
- problemy w których kosztowa funkcja celu zastąpiona została innymi parametrami (np. czas wykonania zleceń, długość tras, ilość przewiezionego ładunku),
- problemy umożliwiające zdefiniowanie kolejności odwiedzania poszczególnych miejsc oraz opcjonalnego odwiedzania niektórych punktów,
- problemy uwzględniające możliwości zwrotów i wysyłki towarów przez klientów (VRP with Backhauls oraz VRP with Pick-Up and Delivery – problem rozwózkowo-zwózkowy),
- problemy, w których warunki zostały ujęte stochastycznie (Stochastic VRP).

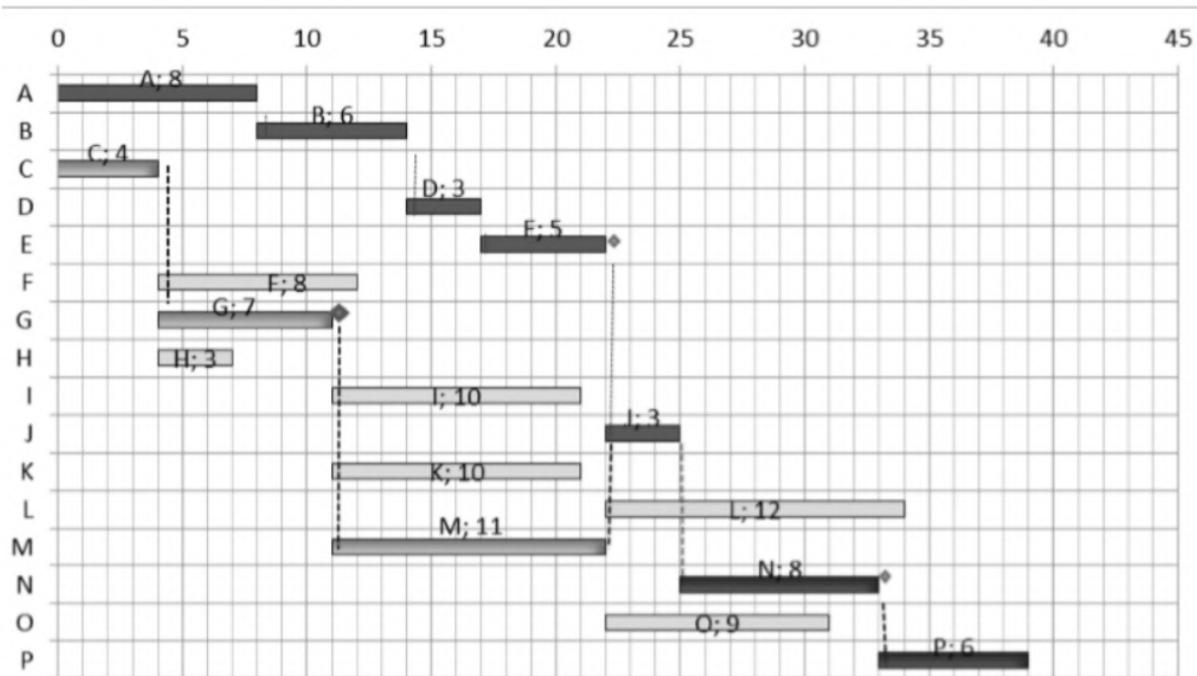
Harmonogram Gantta

CPM

PERT

Harmonogram Gantta

Harmonogram ten można powiedzieć że jest wizualizacją przebiegu projektu. Zaznaczone są na nim wszystkie fazy projektu, czas ich trwania (opcjonalnie możliwy najwcześniejszy/najpóźniejszy termin rozpoczęcia/zakończenia), następstwo czasowe.



◆ kamienie milowe

Metoda ścieżki krytycznej CPM

Podstawą CPM jest stworzenie modelu projektu, który zawiera:

- listę wszystkich zadań wymaganych do realizacji projektu,
- czas trwania każdego z zadań,
- powiązania pomiędzy poszczególnymi czynnościami.

CPM pozwala wyznaczyć:

- ścieżkę krytyczną - najdłuższą ścieżkę działań do zakończenia projektu - ciąg czynności łączących zdarzenia o najmniejszych lub zerowych rezerwach czasu.
- najwcześniejszy i najpóźniejszy termin wystąpienia zdarzenia bez wpływu na długość realizowanego projektu oraz rezerwy czasowe.

Technika ta pozwala na priorytetyzację zadań projektowych poprzez:

- dodanie / podział zadań, które mogą być realizowane równolegle,
- skrócenie czasu trwania zadań ścieżki krytycznej poprzez użycie dodatkowych zasobów.

FAZA I:

- podział projektu na zadania niezbędne do realizacji,
- wyznaczenie powiązań pomiędzy czynnościami.

FAZA II:

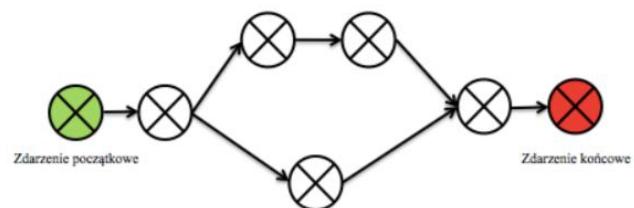
- oszacowanie czasu trwania każdego z zadań,
- wyznaczenie najwcześniejszego i najpóźniejszego terminu wystąpienia zdarzenia oraz rezerw czasu,
- wyznaczenie ścieżki krytycznej,
- przedstawienie struktury projektu w postaci wykresu sieciowego.

FAZA III:

- przedstawienie przebiegu projektu w postaci harmonogramu z uwzględnieniem kamieni milowych,
- planowanie zasobów projektu wraz z wyrównaniem zapotrzebowania na zasoby,
- działania korygujące, które uwzględniają skrócenie czasu trwania projektu.

Zdarzenia:

- początkowe
- końcowe



Terminy:

- Najwcześniejszy Możliwy – NMT, (t_i)
- Najpóźniejszy Dopuszczalny – NDT, (\bar{t}_i)



Rezerwa czasu:

- dla zdarzenia (nazywana **luz**)
- dla czynności (**rezerwa**, zapas)

Procedura 1 : Numerowanie wierzchołków grafu.

CEL: Ponumerować wierzchołki tak, aby zdarzenie poprzedzające miało numer mniejszy niż następujące.

Krok 1: Przydziel wierzchołkowi swobodnemu (nie dochodzą do niego żadne łuki) nr **$i = 1$**

Krok 2: Usuwamy łuki o początku w wierzchołkach ponumerowanych

Krok 3: Wierzchołkom swobodnym przydzielamy kolejne numery **$i+1, i+2 \dots$**

Krok 4: Jeśli nie ponumerowano wszystkich wierzchołków to wykonuj **Krok 2**

Procedura 2 : Obliczanie najwcześniejszych terminów zdarzeń.

Krok 1: Podstaw dla zdarzenia początkowego przedsięwzięcia

$$\underline{t}_1 := 0$$

Krok 2: Dla $j=2,\dots,n$ wykonaj $\underline{t}_j := \max_{i \in W_j^-} \{\underline{t}_i + t_{ij}\}$

Chodzi o to, że jak obliczamy termin najwcześniejszy to obliczamy czasy osiągnięcia zdarzenia przechodząc przez jego „rodziców” (najwcześniejszy termin zdarzenia rodzica + czas czynności pomiędzy rodzicem, a aktualnym) i przypisujemy mu czas najdłuższy. Oznacza to, że zdarzenie może się rozpocząć najwcześniej gdy wszystkie czynności je poprzedzające zostaną zakończone.

Procedura 3 : Obliczanie najpóźniejszych terminów zdarzeń.

Krok 1: Podstaw dla zdarzenia końcowego przedsięwzięcia

$$\overline{t}_n := \underline{t}_n$$

Krok 2: Dla $i=n-1,\dots,1$ wykonaj $\overline{t}_i := \min_{j \in W_i^+} \{\overline{t}_j - t_{ij}\}$

Dla danego wierzchołka wybieramy minimum z czynności rozpoczętujących się w nim i ich czasu najpóźniejszego (suma czynność i czas najpóźniejszy zdarzenia, do którego prowadzi czynność)

Procedura 4 : Obliczanie rezerw i luzów, wyznaczenie ścieżki krytycznej.

Krok 1: Rezerwa czasu dla czynności (i,j) :

$$\forall_{(i,j) \in E} r_{ij} := \overline{t}_j - \underline{t}_i - t_{ij}$$

Jeżeli rezerwa wynosi 0 to czynność leży na ścieżce krytycznej

Krok 2: Luz dla zdarzenia j : $\forall_{j \in V} l_j := \overline{t}_j - \underline{t}_j$

Jeżeli luz wynosi 0 to zdarzenie leży na ścieżce krytycznej.

Warunek zakończenia przedsięwzięcia w terminie:

$$\forall_{j \in V} t_j \in [\underline{t}_j, \overline{t}_j]$$

Modelowanie przedsięwzięcia w programowaniu sieciowym:

- Metoda amerykańska - wykorzystuje sieć zdarzeń
- Metoda francuska (potencjałów) - wykorzystuje sieć czynności

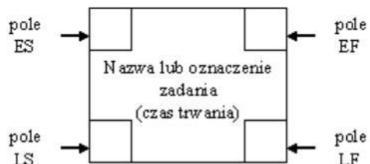
Typy modeli

- Deterministyczny
- Stochastyczny

Wierzchołek reprezentujący zdarzenie:



Wierzchołek reprezentujący czynność:



gdzie:

- ES – early start (lub WS – wcześnie start) – w przód: $ES=EF$ zadania poprzedniego.
- EF – early finish (lub WK – wcześnie koniec) – w przód: $EF=ES$ zadania następnego.
- LS – late start (lub PS – późny start) – wstecz: $LS=LF$ zadania następnego (planowanie „wstecz” = zadanie następne oznacza chronologicznie wcześniejsze).
- LF – late finish (lub PK – późny koniec) – wstecz: $LF=LS$ zadania poprzedniego (jw.)

Algorytm przenumerowania wierzchołków:

Algorytm przenumerowania wierzchołków grafu

Krok 1. Dla danej sieci wyznacz odpowiadającą jej macierz binarną

Krok 2. Do warstwy w_0 zalicz te zdarzenia, które odpowiadają zerowym kolumnom macierzy B

Krok 3. Z macierzy B wykreśl zerowe kolumny oraz wiersze o tych samych numerach co wykreślone kolumny

Krok 4. Do warstwy kolejnej zalicz wierzchołki odpowiadające zerowym kolumnom zredukowanej macierzy B

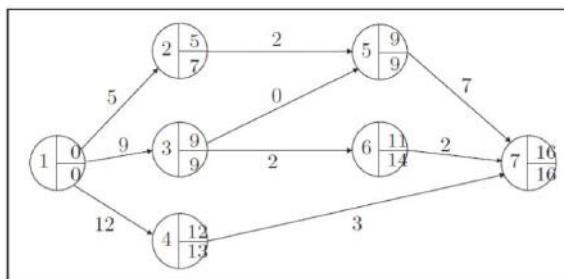
Krok 5. Powtóż czynności 3 i 4

Algorytm przenumerowania wierzchołków grafu

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 3 & 4 & 5 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 4 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

1. Wykreślamy 6 kolumnę i 6 wiersz (warstwa w_0)
2. Wykreślamy 2 kolumnę i 2 wiersz (warstwa w_1)
3. Wykreślamy 3 i 5 kolumnę, 3 i 5 wiersz (warstwa w_2)
4. Wykreślamy 1 kolumnę i 1 wiersz (warstwa w_3)
5. Pozostaje 4 kolumna (warstwa w_4)

Przykład:



- | | |
|--------------------------------------------------|--------------------------------------------------|
| 1. $T_1(w) = 0$ | 8. $T_7(p) = 16$ |
| 2. $T_2(w) = 5 + 0$ | 9. $T_6(p) = 16 - 2 = 14$ |
| 3. $T_3(w) = 9 + 0$ | 10. $T_5(p) = 16 - 7 = 9$ |
| 4. $T_4(w) = 12 + 0$ | 11. $T_4(p) = 16 - 3 = 13$ |
| 5. $T_5(w) = \max\{5 + 2, 9 + 0\} = 9$ | 12. $T_3(p) = \min\{14 - 2, 9 - 0\} = 9$ |
| 6. $T_6(w) = 9 + 2 = 11$ | 13. $T_2(p) = 9 - 2 = 7$ |
| 7. $T_7(w) = \max\{9 + 7, 11 + 2, 12 + 3\} = 16$ | 14. $T_1(p) = \min\{7 - 5, 9 - 9, 13 - 12\} = 0$ |

Ważne:

- czas $\tau = T_n(w) = T_n(p)$ – najkrótszy cykl realizacji przedsięwzięcia
- czynności krytyczne – czynności leżące na ścieżce krytycznej
- ciąg zdarzeń krytycznych nie wyznacza w sposób jednoznaczny ścieżki krytycznej

- Definiujemy zapas całkowity czasu czynności (i, j)

$$z_{ij}(c) = T_j(p) - T_i(w) - t_{ij}$$

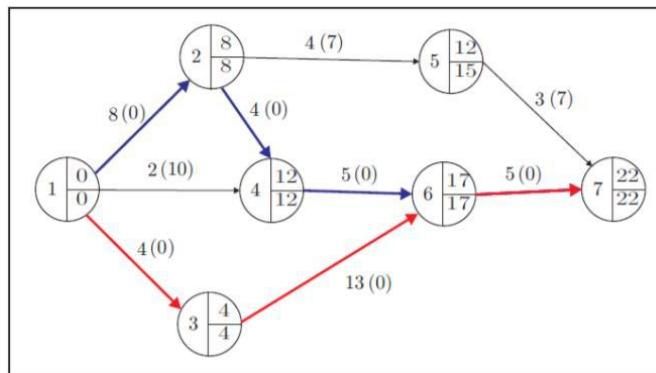
- Zapas całkowity jest rezerwą czasu, którą dana czynność dysponuje wspólnie z innymi czynnościami leżącymi na tym ciągu niekrytycznym

Twierdzenie

Warunkiem koniecznym i dostatecznym na to, aby czynność (i, j) była czynnością krytyczną jest równość

$$z_{ij}(c) = 0$$

Przykład:



Obliczamy zapasy

- | | |
|--------------------------------------------|-------------------------------|
| 1. $z_{12} = T_2(p) - T_1(w) - t_{12} = 0$ | 5. $z_{25} = 19 - 8 - 4 = 7$ |
| 2. $z_{13} = 4 - 0 - 4 = 0$ | 6. $z_{46} = 17 - 12 - 5 = 0$ |
| 3. $z_{14} = 12 - 0 - 2 = 10$ | 7. $z_{36} = 17 - 4 - 13 = 0$ |
| 4. $z_{24} = 12 - 8 - 4 = 0$ | 8. $z_{67} = 22 - 17 - 5 = 0$ |
| | 9. $z_{57} = 22 - 12 - 3 = 7$ |

Zapas całkowity, swobodny, niezależny czasu dla czynności.

- Jeżeli czasy czynności krytycznych nie ulegną wydłużeniu, to całe przedsięwzięcie zostanie zrealizowane w najkrótszym możliwym terminie $T_n(w) = T_n(p)$
- Zapas swobodny czynności (i, j) : $z_{ij}(s) = T_j(w) - T_i(w) - t_{ij}$ określa, o ile jednostek czasu może spóźnić się rozpoczęcie czynności (i, j) , bez naruszenia terminu najwcześniejszego zdarzenia j (bez naruszenia $T_j(w)$)
- Zapas niezależny czynności (i, j) : $z_{ij}(n) = T_j(w) - T_i(p) - t_{ij}$ wyraża dopuszczalne opóźnienie czynności (i, j) w przypadku, gdy zdarzenie i zaistniałoby w terminie najpóźniejszym, a zdarzenie j powinno rozpocząć się w terminie najwcześniejszym
- Pomiędzy typami zapasów zachodzi zależność

$$\begin{array}{ccc} \text{niezależny} & \text{swobodny} & \text{całkowity} \\ z_{ij}(n) & \leq & z_{ij}(s) & \leq & z_{ij}(c) \end{array}$$

- Dla czynności krytycznych wszystkie rodzaje zapasów są równe zero

Harmonogram realizacji przedsięwzięcia:

Na podstawie modelu sieciowego przedsięwzięcia i obliczonych wartościach można zobrazować przedsięwzięcie w postaci diagramu

Niech

- $P_{ij}(w)$ - najwcześniejszy możliwy termin rozpoczęcia czynności (i, j)
- $P_{ij}(p)$ - najpóźniejszy dopuszczalny termin rozpoczęcia czynności (i, j)
- $K_{ij}(w)$ - najwcześniejszy termin zakończenia czynności (i, j)
- $K_{ij}(p)$ - najpóźniejszy dopuszczalny termin zakończenia czynności (i, j)

gdzie:

$$P_{ij}(w) = T_i(w)$$

$$P_{ij}(p) = T_j(p) - t_{ij}$$

$$K_{ij}(w) = T_i(w) + t_{ij}$$

$$K_{ij}(p) = T_j(p)$$

Przykład:

(i, j)	t_{ij}	$P_{ij}(w)$	$P_{ij}(p)$	$K_{ij}(w)$	$K_{ij}(p)$	$Z_{ij}(c)$	$Z_{ij}(s)$	$Z_{ij}(n)$
(1, 2)*	8	0	0	8	8	0	0	0
(1, 3)*	4	0	0	4	4	3	0	0
(1, 4)	2	0	10	2	12	10	10	10
(2, 4)*	4	8	8	12	12	0	0	0
(2, 5)	4	8	15	12	19	7	0	0
(3, 6)*	10	4	4	17	17	0	0	0
(4, 6)*	5	12	12	17	17	0	0	0
(5, 7)	3	12	19	15	22	7	7	0
(6, 7)*	5	17	17	22	22	0	0	0

Metoda PERT – model stochastyczny

PERT (ang. Program Evaluation and Review Technique)

- Sieć o strukturze logicznej zdeterminowanej
- Parametry opisujące poszczególne czynności mają charakter stochastyczny
- Czas trwania każdej czynności jest szacowany:
 - t_c – optymistyczny,
 - t_m - najbardziej prawdopodobny,
 - t_p – pesymistyczny.
- Wyznaczane parametry dla czynności:
 - t_0 – wartość oczekiwana dla rozkładu beta:
$$t_0 = \frac{t_c + 4t_m + t_p}{6}$$
 - σ^2 - wariancja:
$$\sigma^2 = \left(\frac{t_p - t_c}{6}\right)^2$$
 - Prawdopodobieństwo realizacji na podstawie dystrybuanty rozkładu

Przebieg obliczeń dla metody PERT:

1. Definiowanie wszystkich czynności projektu
2. Ustalenie następstwa czasowego czynności
3. Oszacowanie czasu trwania każdej czynności
4. Wyznaczenie ścieżki krytycznej oraz kryteriów jakościowych i ilościowych
5. Tworzenie harmonogramu
6. Przeszacowania i poprawki zgodne ze stanem rzeczywistym

Zasada działania taka sama jak CPM, tylko mamy trzy czasy wejściowe dla czynności t_c, t_m, t_p , na podstawie których wyznaczamy czas trwania czynności. Po wykonaniu obliczeń model pozwala na wyliczenie prawdopodobieństwa, ale trzeba wyznaczyć wariancję:

- Interpretacja wariancji

Im większa jest rozpiętość ocen między czasem optymistycznym i pesymistycznym, tym większa jest niepewność związana z daną czynnością

- Definicja wariancji

$$\sigma^2 = \left(\frac{t_p - t_c}{6} \right)^2$$

Im większa wartość wariancji, tym większa niepewność z czasem trwania danej czynności

Przykład:

Wariant A

(i, j)	t_c	t_m	t_p	t_0	σ^2
*(1, 2)	13	14	15	14	$\frac{1}{9}$
(1, 3)	5	10	15	10	$\frac{25}{9}$
(1, 4)	7	10	19	11	4
*(2, 3)	2	2	2	2	0
(2, 5)	10	10	10	10	0
(3, 6)	20	21	22	21	$\frac{1}{9}$
*(3, 7)	4	16	16	14	4
(4, 7)	5	20	23	18	9
(5, 8)	5	8	11	8	1
(6, 8)	12	12	12	12	0
*(7, 8)	18	18	30	20	4

ścieżka krytyczna:

1 – 2 – 3 – 7 – 8

wariancja całkowita:

$$\sigma^2 = \frac{1}{9} + 0 + 4 + 4 = 8\frac{1}{9}$$

Jak widać interesuje nas wariancja dla ścieżki krytycznej (suma wariancji dla czynności krytycznych). Na podstawie wariancji możemy określić w jakich ramach czasowych zakończy się projekt jako czas obliczony z CPM \pm wariancja ścieżki krytycznej.

Wyznaczanie prawdopodobieństw:

Dane:

σ -wyliczona wariancja

$time$ - wyznaczony czas trwania

- Prawdopodobieństwo dla danego dnia

$$X = \frac{dany_dzień - time}{\sqrt{\sigma}}$$

I z tablic rozkładu normalnego odczytujemy prawdopodobieństwo dla obliczonej wartości

- Dnia z zadanym prawdopodobieństwem

X_- - wartość dla prawdopodobieństwa (odczytana z tablic)

$$dany_{dzień} = X_- \cdot \sqrt{\sigma} + time$$

Wykład 6

Zagadnienie przydziału (AP – Assignment Problem)

Należy wykonać n zadań przy założeniu, że koszt realizacji i -tego zadania przez j -tą maszynę wynosi c_{ij} .

Ograniczenia:

- każda maszyna wykonuje jedno zadanie
- każde zadanie jest wykonywane przez jedną maszynę

Problem polega na minimalizacji kosztów realizacji.

Formalnie:

$$v(AP) = \min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

gdzie:

$x_{ij} \in \{0,1\}$ ~ to znaczy że maszyna może wykonywać dane zadanie albo nie

$\sum_{j=1}^n x_{ij} = 1$ ~ każde zadanie jest wykonywane przez jedną maszynę

$\sum_{i=1}^n x_{ij} = 1$ ~ każda maszyna wykonuje jedno zadanie

Rozwiązywanie **metodą węgierską** (bo zazwyczaj węgierskie nazwiska są trudne do wymówienia) opiera się a dwóch twierdzeniach

Twierdzenie o redukcji macierzy (na wykładzie nazwane po prostu TW.1.)

- przydatne dla AP oraz TSP
- rozwiązywanie problemu nie zmieni się jeżeli od kolumny/wiersza macierzy odejmiemy stałą wartość
- jeżeli x jest rozwiązaniem problemu z macierzą A to jest ono też rozwiązaniem macierzy zredukowanej

TW.2.

Dla macierzy o nieujemnych elementach, jeżeli rozwiązanie $v(AP) = 0$ (dla macierzy zredukowanej w szczególności) to jest to rozwiązanie optymalne.

Zero niezależne w macierzy problemu AP to takie, że żadne dwa zera niezależne nie leżą w jednej kolumnie lub wierszu. Można je oznaczyć jako 0^* .

Metoda węgierska, algorytm węgierski, algorytm Kuhna-Munkersa:

- krok przygotowawczy:

redukujemy macierz odejmując od każdej kolumny najpierw najmniejsze elementy wiersza a potem kolumny a suma odjętych elementów to nowe dolne ograniczenie dla funkcji celu.

-krok 2:

poszukujemy kompletnego przydziału (zbioru zer niezależnych o mocy n tj. rozmiaru problemu)

jeżeli się powiodło to kompletny przydział jest rozwiązaniem o koszcie dolnego ograniczenia funkcji celu.

jeżeli się nie powiodło to wyznaczamy wszystkie zera zależne

-krok 3:

wykreślenie wszystkich zer macierzy minimalną liczbą linii pionowych lub poziomych

jeżeli liczba linii to wielkość problemu to znaleźliśmy rozwiązanie

w innym wypadku: krok 4

-krok 4:

szukamy najmniejszy nieprzykryty przez linie element macierzy A: x

odejmujemy od nieprzykrytych liniami elementów x

do przykrytych dwoma liniami elementów A dodaj x

do dolnego ograniczenia funkcji kosztu dodaj x

Oczywiście wykreślenie zer minimalną liczbą zer wymaga nieco więcej gimnastyki:

1. Poszukiwanie maksymalnego skojarzenia:

-zaznaczanie każdego wiersza nie posiadającego zera niezależnego.

-zaznaczenie każdej kolumny z zerem zależnym

- zaznaczenie każdego wiersza mającego w zaznaczonej kolumnie niewiążące zero

Robimy to do oporu/odcięcia/nie wiem jak się teraz mówi: dopóki nie jest możliwe dalsze oznakowanie.

2. Poszukujemy minimalnego pokrycia wierzchołkowego:

pokrycie wierzchołkowe to zbiór minimalny wierszy i kolumn zawierający wszystkie zera w macierzy.

jest on wyznaczony przez przekreślenie wszystkich nieoznakowanych wierszy i oznakowanych kolumn.

Problemy szeregowania zadań (do Johnson dla 2 maszyn)

Po co? → Do optymalizacji produkcji na wielu płaszczyznach działalności w przemyśle etc.:

- planowanie produkcji
- proces wytwarzania
- dostarczanie produktu do odbiorcy

Przez postęp i bardziej złożone systemy produkcyjne i informatyczne problemy szeregowania składają się z **coraz większej liczby maszyn** (procesów). Niesie to ze sobą wzrost liczby **realizowanych zadań** co przekłada się na potrzebę optymalizacji coraz lepszymi metodami.

W większości przypadków takie problemy cechują się **złożonością wielomianową** przez co lepiej (znany sposób) jest wykorzystać **algorytmy przybliżone** - taki kompromisik pomiędzy **jakością** uzyskanego rozwiązania a **czasem obliczeń**.

Samo szeregowanie to tworzenie **harmonogramów zadań** (czynności, programów, produkcji etc) na maszynach (procesory, stanowiska obsługi, obrabiarki wiertarki wszystko co może wykonać zadanie).

Szukanie harmonogramu dla danego zbioru zadań w określonych warunkach ma na celu **zminimalizowanie** przyjętego **kryterium uszeregowowania**.

Założenia przy tworzeniu harmonogramów (w ogólności):

- pomijamy czas i koszt transportu
- maszyna może wykonywać co najwyżej 1 operację naraz
- operacje wykonywane są bez przerwy (o ile to możliwe)

Podstawowe pojęcia:

- podstawowe elementy modeli szeregowania to dwa niepuste zbiory: zbiór **maszyn** i zbiór **zadań**
- każde zadanie przeznaczone jest do wykonania na co najmniej jednej maszynie (z uwzględnieniem specyficznych dla danego problemu założeń)
- **deterministyczne problemy szeregowania zadań:** takie problemy, w których parametry liczbowe opisujące dany system są znane - podstawowa grupa w teorii szeregowania zadań

- **problemy probabilistyczne:** momenty przybycia zadań oraz czasy ich obsługi nie są dokładnie znane, natomiast znane są ich rozkłady prawdopodobieństwa

Typy maszyn:

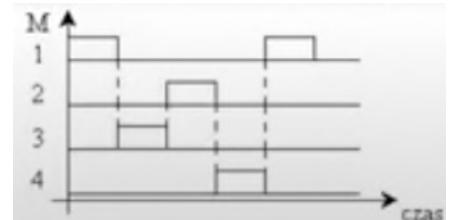
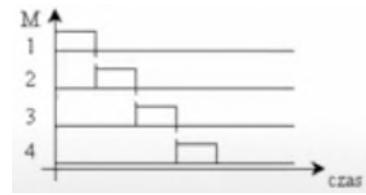
- **równolegle:** maszyny realizujące identyczne funkcje (np. wszystkie maszyny którymi możemy zrobić jakąś czynność: np wiertarką, frezarką i dławem możemy zrobić dziurę)
- **dedykowane:** maszyny realizujące różne funkcje (mogą wykonać tylko jakieś jedno specyficzne zadanie, jak np. powiedzmy maszynka która potrafi tylko i wyłącznie obrać jabłko i nic więcej, i żadna inna maszynka tego nie potrafi zrobić)

W systemach z obsługą odbywającą się na maszynach **równoległych** wyodrębniamy następujące rodzaje maszyn:

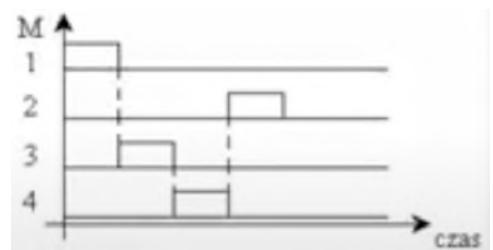
- *identyczne (P)* - jednakowa prędkość wykonywania zadań
- *jednorodne (Q)* - różnią się szybkością przetwarzania
- *niezależne (R)* - dowolna szybkość wykonywania zadań (możemy ją sobie ustawić)

W systemach z maszynami **dedykowanymi** wyróżniamy 3 typy szeregowania:

- **flow-shop** - system przepływowowy gdzie zadania wykonywane są przez wszystkie maszyny w **identycznej kolejności** → pojedyncze zadanie składa się z **operacji** które są realizowane na odrębnych maszynach zgodnie z narzuconymi ograniczeniami kolejnościowymi (można sobie wyobrazić linię produkcyjną z jednym taśmociągiem)
- **job-shop** - system gniazdowy, w którym zadania wykonywane są w **dowolnej kolejności** przez dowolny podzbiór maszyn, przy czym kolejność wykonania danego zadania na przydzielonym podzbiorze maszyn jest uprzednio określona - kolejność realizacji operacji w ramach danego zadania jest narzucona (kolejność zadań zachowana, ale kolejność maszyn nie)



- **open-shop** - system otwarty, w którym zadania są wykonywane przez wszystkie maszyny w dowolnej kolejności - brak relacji kolejnościowych dla operacji poszczególnych zadań - zupełnie dowolnie, ważne żeby wszystkie były zrobione



Notacja w zapisie problemów:

Informacja o maszynach | informacja o zadaniach | funkcja celu

$$\alpha | \beta | \gamma$$

Jeżeli którejś informacji brakuje, kreski nadal zostają, tj. jeżeli mamy informacje o maszynach, funkcji celu ale nie o zadaniach zapis wyglądałby tak: $\alpha | | \gamma$

Znaczenie symboli:

$$\alpha = \alpha_1 \alpha_2$$

$\alpha_1 \in \{\emptyset, P, Q, R\}$ → dla maszyn **równoległy**

- \emptyset → jedna maszyna
- P → identyczne (równe czasy wykonania zadań, dokładniejszy opis wyżej)
- Q → jednorodne (różnią się szybkością przetwarzania)
- R → niezależne (dowolna prędkość przetwarzania)

$\alpha_1 \in \{F, O, J\}$ → dla maszyn **dedykowanych** (Flow-shop, Open-shop, Job-shop)

α_2 – liczba maszyn (m)

Czyli np. zapis $\alpha = J4$ oznacza problem job-shop z czterema maszynami.

$$\beta = \beta_1 \beta_2 \dots$$

$\beta_1 \in \{\emptyset, podzielny\}$ → podzielność oznacza że zadanie można podzielić (przerwać) skońzoną ilość razy, np. jeżeli zadanie to ‘zrób 500 śrubek’ to można je podzielić na 100 i 400 śrubek.

$\beta_2 \in \{\emptyset, r_j\} \rightarrow$ termin przybycia zadań może być różny (release time - r_j) takie odzwierciedlenie np transportu materiałów do fabryki (dojadą za X, więc przed X nie zrobimy zadania jakiegoś). Dodatkowo potencjalnie możemy sobie zdefiniować np. termin zakończenia d_j - termin którego nie można przekroczyć lub trzeba przekroczyć go jak najmniej oraz także wagę w_j używaną później do funkcji celu (ostatnia na liście poniżej)

γ - funkcja celu

c_j - czas zakończenia zadania j-ego

Różne warianty, tutaj parę z nich:

$c_{max} = \max c_j \rightarrow$ minimalny termin najpóźniej zakońzonego zadania

$\sum_{j=1}^n c_j \rightarrow$ sumaryczny czas zakończenia

$\frac{1}{n} \sum_{j=1}^n c_j \rightarrow$ średni czas zakończenia zadania

$\sum_{j=1}^n (c_j - r_j) \rightarrow$ czas przebywania zadań w systemie

$\sum_{j=1}^n w_j c_j \rightarrow$ jakąś przez nas zdefiniowana suma ważona (większa waga ważniejsze zadanie)

Definicja Harmonogramu dopuszczalnego:

Harmonogram nazywamy dopuszczalnym jeżeli przyporządkowanie maszyn ze zbioru M do zadań ze zbioru J spełnia następujące warunki:

1. maszyna wykonuje co najwyżej 1 zadanie naraz
2. każde zadanie jest realizowane w przedziale czasu $[r_j, \inf)$ począwszy od chwili przybycia r_j
3. wszystkie zadania z J są wykonane
4. jeżeli zadanie i poprzedza zadanie j , to zadanie j może się rozpocząć po wykonaniu zadania i
5. w przypadku zadań niepodzielnych wykonanie nie może być przerwane - jeżeli dopuszcza się przerwanie, to liczba przerwań musi być skończona

Definicja Uszeregowania optymalnego:

Jest optymalne jeżeli minimalizuje lub maksymalizuje określone kryterium szeregowania (funkcję celu gamma)

Przykład: klasyczny flow-shop nazywany także permutacyjnym problemem przepływowym zapisywany najczęściej jest jako $F||C_{\max}$, jeżeli optymalizowana wartość to długość uszeregowania (termin zakończenia)

Dany jest zbiór zadań: $J = \{J_1, J_2, \dots, J_n\}$
oraz maszyn: $M = \{M_1, M_2, \dots, M_m\}$.
Ciąg operacji stanowiący zadanie j : $(O_{1j}, O_{2j}, \dots, O_{mj})$

Operacja O odpowiada zadaniu $j \in J$ wykonywanemu na maszynie i w ścisłe określonym czasie p_{ij} .

Zadania ze zbioru J wykonywane są na maszynach ze zbioru M z uwzględnieniem następujących założeń:

- w danej jednostce czasu każda z maszyn może wykonywać co najwyżej jedno zadanie,
- rozpoczęta operacja nie może zostać przerwana,
- wykonywanie zadania na maszynie i może zostać rozpoczęte wyłącznie po zakończeniu na maszynie $i-1$.

Rozwiążanie tak postawionego zadania jest w pełni scharakteryzowane przez串 terminów rozpoczęcia $S = (S_1, \dots, S_n)$, gdzie $S_j = (S_{1j}, \dots, S_{mj})$ lub innymi串 zakończenia zadań $C = (C_1, \dots, C_n)$, gdzie $C_j = (C_{1j}, \dots, C_{mj})$.

Oznaczając przez π pewną permutację $(\pi(1), \pi(2), \dots, \pi(n))$ zbioru zadań oraz zbiór wszystkich permutacji zbioru J przez Π , zadaniem optymalizacji jest znalezienie takiej permutacji $\pi^* \in \Pi$, dla której:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi),$$

gdzie $C_{\max}(\pi) = C_{m\pi(n)}$. Terminy zakończenia poszczególnych zadań i ostatecznie串 długość uszeregowania $C_{\max}(\pi)$ obliczana jest przyjmując zerowe warunki początkowe $\pi(0) = 0$, $C_{i0} = 0$ i $C_{0j} = 0$, ze wzoru rekurencyjnego:

$$C_{i\pi(j)} = \max\{C_{i,\pi(j-1)}, C_{i-1,\pi(j)}\} + p_{i\pi(j)}, \quad i = 1, \dots, m, \quad j = 1, \dots, n,$$

Linijka wyżej warto wiedzieć: $\max\{\text{czas zwolnienia poprzedniej maszyny, czas zwolnienie poprzedniego zadania}\} + \text{czas wykonywania danego zadania}$

Reprezentacja danych:

Zazwyczaj jest to macierz czasów realizacji kolejnych operacji o liczbie wierszy równej liczbie maszyn (m) oraz liczbie kolumn równej liczbie zadań (n)

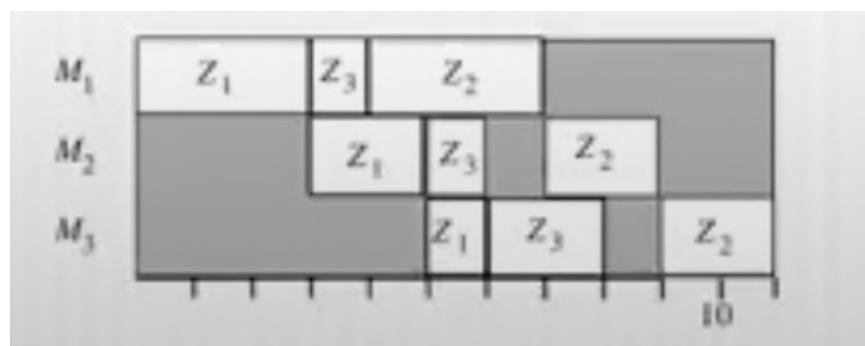
		columns			
		p_{11}	\dots	p_{1j}	\dots
p_{i1}	\dots	p_{ij}	\dots	p_{in}	
	\dots	p_{mj}	\dots	p_{mn}	
p_{m1}	\dots				

Przykład rozwiązywanych harmonogramów (szare przerwy to przestoje maszyn: czasy pomiędzy realizacją Z_n na poszczególnych maszynach to czas oczekiwania zadania)

- open-shop:



- flow-shop:



Algorytmy do rozwiązywania tych arcy ciekawych zagadnień

Przypadek dwóch lub trzech maszyn:

- algorytm Johnsona (dokładny)

Dla większej liczby maszyn (algorytmy przybliżone):

- Campbella-Dudka-Smitha (CDS)
- Browna-Łomnickiego

Algorytm Johnsona działanie (dla dwóch maszyn):

Oznaczenia:

t_{ij} – czas obróbki na i -tej maszynie j -tego detalu

T_{ij} – czas zakończenia obróbki na i -tej maszynie j -tego detalu

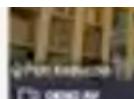
$$T_{1j} = \sum_{k=1}^J t_{1k} \text{ (pierwsza maszyna } j\text{-te zadanie)}$$

$$T_{n1} = \sum_{l=1}^n t_{l1} \text{ (n-ta maszyna 1-sze zadanie)}$$

$$T_{ij} = t_{ij} + \max(T_{i,j-1}, T_{i-1,j})$$

Schemat działania: (ponizej powinno być 2xn)

Dana jest macierz $2 \times m$ czasów operacji t_{ij}



- 1) Znajdź najmniejszy element macierzy t_{ij} ($\min_{i,j} t_{ij}$)
- 2) Jeśli ten element znajduje się
 - w pierwszym wierszu ($\min t_{ij} = t_{1k}$), to optymalna kolejność obróbki musi się rozpocząć detalem o nr k ,
 - w drugim wierszu ($\min t_{ij} = t_{2s}$) to optymalna kolejność kończy się detalem o numerze s .
- 3) Po ustaleniu kolejności w kroku 2, skreślamy w macierzy czasów odpowiadającą mu kolumnę i powtarzamy to postępowanie.
Wybrany kolejny detal ustawiamy na pierwszym wolnym miejscu (licząc od początku lub końca).
Idź do → (1)

Algorytm kontynuujemy, aż ustalimy kolejność wszystkich detali.

Fragment działania algorytmu:

	Z1	Z2	Z3	Z4	Z5	Z6
M1	9	6	8	7	12	3
M2	7	3	5	10	4	7
3. Kolejny element minimalny znajduje się w drugim wierszu (Z1 kończy „wolny” harmonogram) i pierwszym (Z4 ustawia się na pierwszą wolną pozycję)						
	Z6	Z4	Z1	Z3	Z5	Z2
M1	3	7	9	8	12	6
M2	7	10	7	5	4	3

Kiedy to już po wykreśleniu Z2, Z3, Z5 oraz Z6 i ułożenia ich odpowiednio w wynikowej macierzy zostaje nam Z1 oraz Z4. Jako że minimum z nie wykreślonych kolumn =7, jedno w dolnym jedno w górnym wierszu to nie ma problemu - Z4 (minimum w górnym wierszu - M1) przenosimy w pierwsze wolne miejsce macierzy wynikowej (licząc od lewej) natomast Z1 (minimum w wierszu M2) na pierwsze wolne miejsce licząc od prawej. Jako że uporządkowaliśmy tak wszystkie zadania algorytm kończy swoje działanie.

Wykład 8

Szeregowanie zadań

Zakładamy, że mamy na m ilości urządzeń rozwiązać n czynności. Optymalne szeregowanie to takie, że wykonujemy czynności w określonej kolejności na każdej kolejnej maszynie tak, aby zajęło to jak najmniej czasu.

Założenia:

Każda z m maszyn wykonuje zadanie w tej samej kolejności. Algorytm zakłada dobór tej kolejności. Zadanie musi być wykonane przez wszystkie dostępne maszyny w odgórnie założonej kolejności maszyn. Maszyny nie mogą jednocześnie pracować nad tym samym zadaniem.

Algorytm Johnsona (dokładny) dla dwóch maszyn (złożoność $O(n^*logn)$) :

Schemat postępowania:

Szukamy zadania z najkrótszym czasem wykonywania. Jeżeli czas jest dla maszyny nr 1, to czynność wykonywana jest jako pierwsza, natomiast jeśli jest to czas dla maszyny nr 2, to czynność jest wykonywana jako ostatnia.

Algorytm powtarzamy. Dla kolejnego najmniejszego czasu, dla maszyny nr 1 zadanie trafia na początkową część kolejki (ALE: jako kolejna czynność! Niekoniecznie jako pierwsza), jeśli dla maszyny nr 2 to zadanie trafia na końcową część kolejki (ALE: nie trafia bezpośrednio na sam koniec jeśli już się tam coś znajduje).

Schemat powtarzamy rozpatrując każdą kolejną czynność.

Szukamy czasu zakończenia zadania: do sumy ogólnej dodajemy zsumowane czasy wykonywania czynności przez maszynę nr 1. Patrząc na obłożenie czasowe czynności wykonywanych przez maszynę nr 2, pilnujemy aby wybrane zadanie nie było wykonywane jednocześnie przez dwie maszyny. Dlatego dopuszczać możliwość czekania maszyny 2. (np. Na początku gdy maszyna 1 pracuje nad zadaniem pierwszym z wyznaczonej optymalnej kolejności). Do sumy ogólnej dodajemy to ile maszyna nr 2 pracuje dłużej niż maszyna nr 1.

Przykład: (6 zadań, 2 maszyny, kolejność działania maszyn: M1, M2)

	Z1	Z2	Z3	Z4	Z5	Z6
M1	9	6	8	7	12	3
M2	7	3	5	10	4	7

Wybieramy element minimalny (Z2 dla M2, Z6 dla M1)

Kolejka = [Z6, _, _, _, _, Z2]

	Z1	Z2	Z3	Z4	Z5	Z6
M1	9	6	8	7	12	3
M2	7	3	5	10	4	7

Wybieramy element minimalny (Z5 dla M2)

Kolejka = [Z6, _, _, _, Z5, Z2]

	Z1	Z2	Z3	Z4	Z5	Z6
M1	9	6	8	7	12	3
M2	7	3	5	10	4	7

Wybieramy element minimalny (Z3 dla M2)

Kolejka = [Z6, _, _, Z3, Z5, Z2]

	Z1	Z2	Z3	Z4	Z5	Z6
M1	9	6	8	7	12	3
M2	7	3	5	10	4	7

Wybieramy element minimalny (Z1 dla M2, Z4 dla M1)

Kolejka = [Z6, Z4, Z1, Z3, Z5, Z2]

Powyższa kolejność jest najbardziej optymalna czasowo.

Czas wykonywania:

suma = $3+7+9+8+12+6 = 45$ – zsumowany czas maszyny dla M1

suma += 3 – tyle dłużej pracuje M2 z uwzględnieniem czekania na zadanie.

Suma = 48

Algorytm Johnsona (dokładny) dla trzech maszyn:

Schemat postępowania:

Rozwiązujeśmy przez potraktowanie jak dla dwóch maszyn, przy użyciu transformaty:

$$t'_{1j} = t_{1j} + t_{2j}$$

$$t'_{2j} = t_{2j} + t_{3j}$$

Tzn. czas dla każdego zadania to:

Tj dla M1 to suma wcześniejszego Tj dla M1 i Tj dla M2

Tj dla M1 to suma wcześniejszego Tj dla M2 i Tj dla M3

WARUNEK:

$$\min_j t_{1j} \geq \max_j t_{2j}, \quad j = 1, 2, \dots, m$$

lub

$$\min_j t_{3j} \geq \max_j t_{2j}, \quad j = 1, 2, \dots, m$$

najmniejszy koszt dla maszyny 1 musi być większy lub równy największemu kosztowi maszyny 2

lub

najmniejszy koszt dla maszyny 3 musi być większy lub równy największemu kosztowi maszyny 2

Długość uszeregowania wyznaczmy na podstawie macierzy czasowej dla trzech maszyn. Sumujemy czasy dla maszyny nr 1. Dodajemy do tego czas będący różnicą czasu dla M2 – M1 (z uwzględnieniem przerw na czekanie. Przypomnienie: dwie maszyny nie mogą równocześnie pracować nad jednym zadaniem). Dodajemy do tego czas M3 – M2 (uwzględniają przerwy na czekanie)

Przykład: (5 zadań, 3 maszyny)

	Z1	Z2	Z3	Z4	Z5
M1	7	11	8	7	6
M2	6	5	3	5	3
M3	4	12	7	8	3

Sprawdzamy warunek:

Najmniejsza wartość dla M1 – 6

Największa wartość dla M2 – 6

$6 \leq 6$

Warunek spełniony, nie musimy sprawdzać alternatywnego warunku. Możemy rozwiązywać zadanie jak dla dwóch maszyn

	Z1	Z2	Z3	Z4	Z5
M1	7	11	8	7	6
M2	6	5	3	5	3
M3	4	12	7	8	3
$t_{1j} + t_{2j}$	13	16	11	12	9
$t_{2j} + t_{3j}$	10	17	10	13	6

Schemat wyznaczania kolejności uszeregowania w przykładzie powyżej.

Kolejka jaką otrzymujemy to:

Kolejka1 = [Z4, Z2, Z1, Z3, Z5]

lub

Kolejka2 = [Z4, Z2, Z3, Z1, Z5]

Obie wersje są poprane, zajmują tyle samo czasu.

Czas wykonywania (wyznaczony dla Kolejka1)

Suma = 7+11+8+7+6 = 39 – zsumowany czas dla M1

Suma += 3 – M2-M1 z uwzględnieniem czekania

Suma += 7 - M3-M2 z uwzględnieniem czekania

Suma = 49

Algorytm CDS (Campbella-Dudka-Smitha) – algorytm przybliżony:

Schemat postępowania:

Algorytm wykonuje się na bazie algorytmu Johnsona dla dwóch maszyn. Cały problem dzielimy na mniejsze podproblemy. Zgodnie z zależnością:

$$M_{1j}^r = \sum_{i=1}^r t_{ij} \rightarrow \text{odpowiada } t_{1j}$$
$$M_{2j}^r = \sum_{i=n+1-r}^n t_{ij}, \rightarrow \text{odpowiada } t_{2j}, \quad r = 1, 2, \dots, \text{maszyna}-1, \quad j - \text{zadania}$$

Każdy z podproblemów rozwiążemy osobno, wg algorytmu Johnsona. W ten sposób otrzymujemy kilka wyników dla czasu wykonywania. Odpowiedzią jest ten najmniejszy.

Przykład (5 zadań, 4 maszyny):

Problem podany poniżej dzielimy na podproblemy:

	Z1	Z2	Z3	Z4	Z5
M1	12	7	10	4	16
M2	10	12	6	15	8
M3	6	18	8	13	6
M4	15	9	12	7	10

Podproblem 1:

M1 := M1

M2 := M4

M¹_{ij}	Z1	Z2	Z3	Z4	Z5
M1 (t_{1j})	12	7	10	4	16
M2 (t_{4j})	15	9	12	7	10

Czas wykonania to 96 (rozwiązujeąc podproblem za pomocą algorytmu Johnsona)

Podproblem 2:

$$M1 := M1 + M2$$

$$M2 := M4 + M3$$

M¹_{ij}	Z1	Z2	Z3	Z4	Z5
M1 ($t_{1j} + t_{2j}$)	22	19	16	19	24
M2 ($t_{4j} + t_{3j}$)	21	27	20	20	16

Czas wykonania to 92 (rozwiązujeąc podproblem za pomocą algorytmu Johnsona)

Podproblem 3:

$$M1 := M1 + M2 + M3$$

$$M2 := M4 + M3 + M2$$

M¹_{ij}	Z1	Z2	Z3	Z4	Z5
M1 ($t_{1j} + t_{2j} + t_{2j}$)	28	37	24	32	30
M2 ($t_{4j} + t_{3j} + t_{2j}$)	31	39	26	35	24

Czas wykonania to 96 (rozwiązujeąc podproblem za pomocą algorytmu Johnsona)

Czasy wykonywania podproblemów to kolejno: 96, 92, 96. Tak więc odpowiedź to 92.

Algorytm Browna-Łomnickiego – algorytm przybliżony:

Schemat postępowania:

W celu rozwiązania zadania posługujemy się wzorem:

$$g^i = T(i, w) + \sum_{j \notin w} t_{ij} + \min_{j \notin w} \sum_{l=i+1}^n t_{lj}, \quad i = 1, 2, \dots, n-1$$

.....

$$g^n = T(n, w) + \sum_{j \notin w} t_{nj}$$

Gdzie:

g_i – czas zakończenia zadań W na i tym stanowisku czyli: suma czasów wykonywania zadań na stanowisku oraz suma najkrótszych czasów potrzebnych do wykonania zadań na innych maszynach

g_n – czas zakończenia zadania przez ostatnią maszynę, czyli suma czasów trwania wykonywania zadań przez konkretną maszynę

Założenia: stosujemy we wszystkich przypadkach, gdzie liczba maszyn ≥ 3

Przykład:

$n \setminus m$	1	2	3	4	5
1	10	5	6	5	14
2	12	7	10	7	9
3	14	12	10	12	8
4	6	10	8	4	14
5	8	4	12	3	10

W niech będzie jednoelementowym zbiorem
 $W = \{1\}$

$$G^1 = 10 + (5+6+5+14) + \min[(7+12+10+4), (10+10+8+12), (7+12+4+3), (9+8+14+10)] = 66$$

$$G^2 = 22 + (7+10+7+9) + \min[(12+10+4), (10+8+12), (8+14+10)] = 74$$

$$G^3 = 85$$

$$G^4 = 81$$

$$G^5 = 79$$

$$\text{Max}(G_1, \dots, G_5) = 85$$

n – stanowiska/maszyny

m – detale/operacje/zadania

Wykład 9

Programowanie dynamiczne jest to metoda:

- wieloetapowego rozwiązywania zagadnień decyzyjnych
- poszukująca optymalnej strategii (planu działania, sposób prowadzenia procesu)

Proces decyzyjny może być wieloetapowy z natury (czas) lub zdekomponowany na etapy w sposób sztuczny. W każdym etapie podejmowana jest **decyzja** (zmienna, wektor). **Stan procesu** (zmienna, wektor) jest rozpatrywany przed i po każdym etapie, zmienia się w wyniku podejmowanych decyzji (**funkcji przejścia**).

Zastosowania PD:

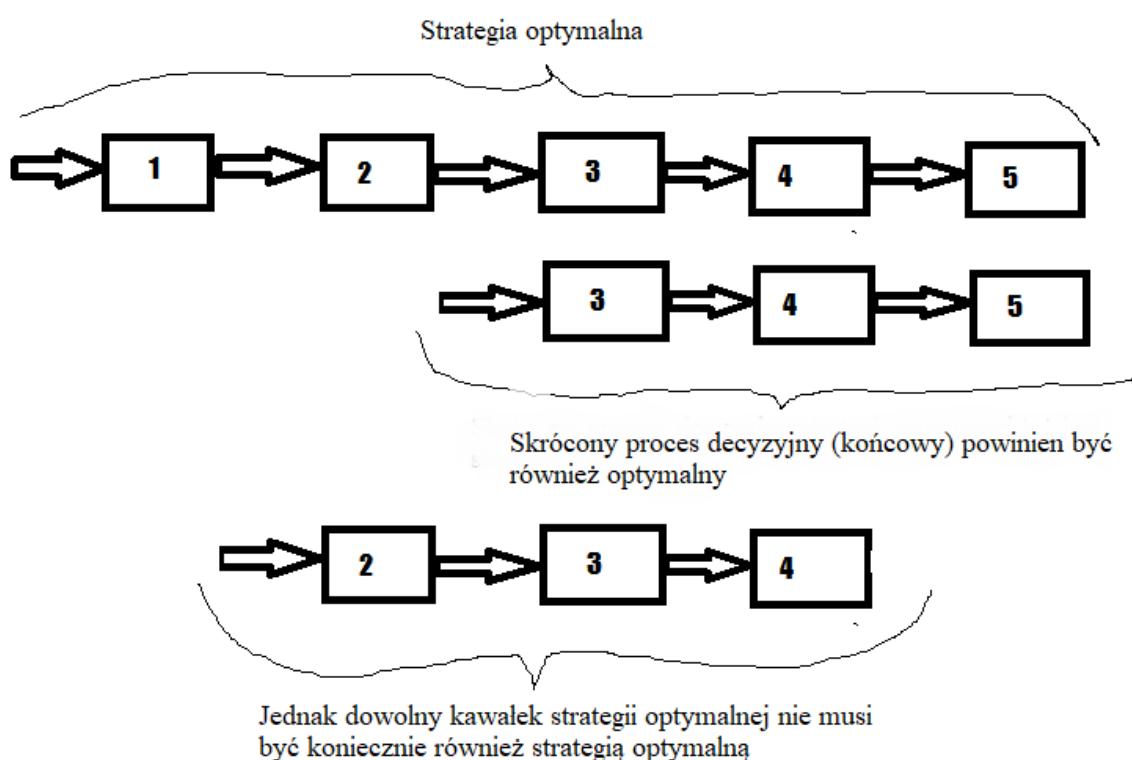
- Zagadnienia statyczne lub dynamiczne
- Modele deterministyczne i stochastyczne
- Zmienne dyskretnie i ciągłe
- Skończony i nieskończony horyzont decyzji

Zasada optymalności w ujęciu Bellmana:

Optymalna strategia sterowania ma tę własność, że jakikolwiek by był stan początkowy i decyzja początkowa, to następne decyzje muszą tworzyć optymalną strategię sterowania względem stanu wynikającego z pierwszej.

Wnioski z powyższego twierdzenia:

- Każdy końcowy odcinek strategii optymalnej jest dla swoich warunków początkowych strategią optymalną,
- **Nie jest prawdą**, że dowolny odcinek strategii optymalnej jest strategią optymalną



Trochę teorii w przedmiocie teoria sterowania z przyszłego semestru:

Układ dynamiczny jest opisany za pomocą równania stanu tzn. istnieje funkcja transzycji stanu:

$$x(t) = \Phi(t, t_0, x(t_0), u) \text{ dla wszystkich } t_0 < t,$$

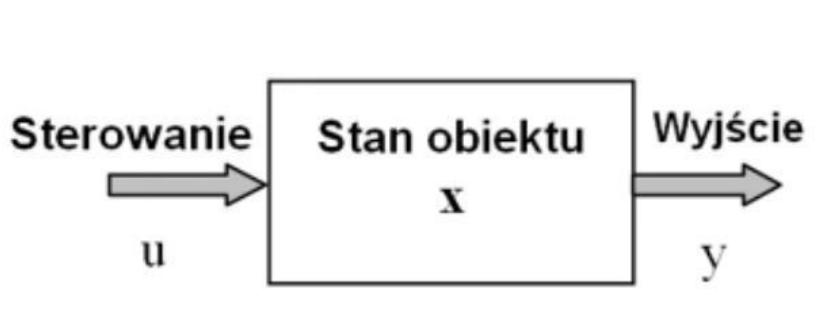
Gdzie x należy do przestrzeni funkcji stanu (x -stan), u należy do przestrzeni funkcji sterowań (u – sterowanie).

Mówiąc w skrócie ważnym jest dla nas samo to że istnieje taka funkcja, która pozwoli nam wyznaczyć jaki stan osiągniemy w czasie t , przy założeniu, że mamy sterowanie, czas, czas początkowy i stan z czasu początkowego.

Funkcjonał jakości jest separowany w czasie tzn. jeśli $t_0 = \inf t, t_N = \sup t$ to istnieją funkcjonały Q' oraz Q'' , takie że:

$$Q(x''', u''') = Q'(x'[t_0, t_1], u'[t_0, t_1]) + Q'(x'[t_0, t_1], u'[t_0, t_1])$$

Możemy pewną sekwencję decyzji przedzielić w dowolnym momencie (np. na etap pierwszy oraz całą resztę). Da nam to funkcjonał o danym stanie i sterowaniu. Będzie to często wykorzystywana zależność.



Zakładamy, że wyjście y jest identyczne ze stanem x : $y \equiv x$. Niech stan obiektu opisany będzie następującym równaniem różnicowym:

$$x_{i+1} = f_i(x_i, u_i)$$

Gdzie i oznacza zmienną niezależną (dyskretyzowany czas).

Jeśli wyjście jest identyczne ze stanem, to kolejny stan obiektu jest zależny od stanu poprzedzającego oraz sterowania, które mamy.

Stan początkowy jest znany. Powyższe równanie opisuje własności dynamiczne obiektu. Równanie to obowiązuje dla każdego i . Można więc zapisać, że:

$$x_1 = f_0(x_0, u_0), \quad x_2 = f_1(f_0(x_0, u_0), u_1) \text{ itd.}$$

Jak widać stan pierwszy zależy od stanu początkowego oraz sterowania w tej chwili, natomiast stan drugi zależy od funkcji ze stanu pierwszego i sterowania. Taka analogia i zagnieżdżanie funkcji ma miejsce w każdym z N stanów.

Mamy również pewien wskaźnik jakości który sumujemy po kolejnych etapach:

$$J = L_0(x_0) + \sum_{i=1}^N L_i(x_i, u_{i-1})$$

Należy znaleźć N-elementowy ciąg sterowań (u_0, \dots, u_{N-1}) minimalizuje (lub maksymalizuje zależnie od rozważanego przypadku) wskaźnik jakości J. Dopuszcza się również dodatkowe ograniczanie w postaci funkcji wektorowej g_i :

$$g_i(x_{i-1}, u_{i-1}) \leq d_i \quad \text{dla } i = 1, \dots, N$$

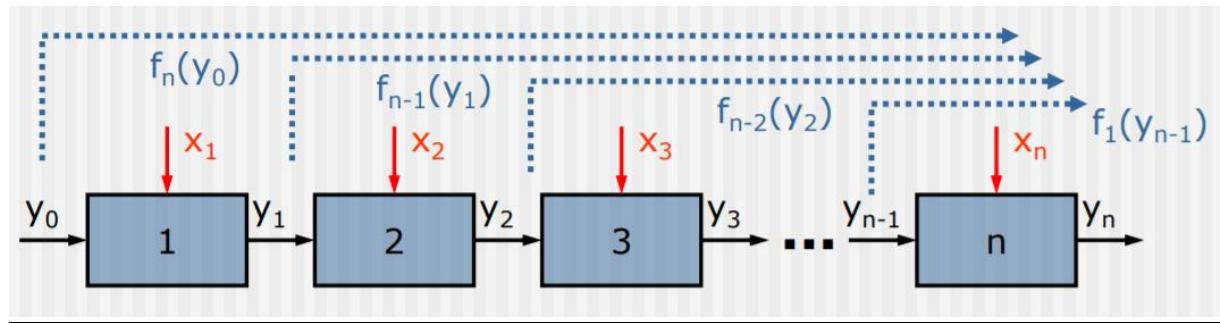
No i tym sposobem dążymy do optymalności, a dokładniej do optymalnego sterowania, którą dla kroku N określamy ogólnie w następujący sposób (z oznaczeniami opisanymi wcześniejszej):

$$G_N(x_{N-1}) = \min L_N(f_{N-1}(x_{N-1}, u_{N-1}), u_{N-1})$$

Poszukując optymalnego sterowania dla $i = N-2$, można wykorzystać otrzymany przed chwilą wynik:

$$G_{N-1} = \min [L_{N-1}(x_{N-1}, u_{N-2}) + G_N(x_{N-1})]$$

Schemat wieloetapowego procesu decyzyjnego:



Oznaczenia (różniące się od tych wcześniejszych), które będą obowiązywać we wszystkich kolejnych przykładach:

- $i = 1, 2, \dots, n$ – etapy procesu decyzyjnego
- n – liczba etapów procesu decyzyjnego
- y_{i-1} – stan wejściowy i-tego etapu (stan przed etapem)
- x_i – decyzja podjęta w i-tym etapie - $x_i \in X_i$ (zbiór decyzji dopuszczalnych etapu)
- y_i – stan wyjściowy i-tego etapu procesu decyzyjnego - $y_i \in Y_i$ (zbiór stanów dopuszczalnych etapu)
- y_0 – stan początkowy procesu decyzyjnego
- y_n – stan końcowy procesu decyzyjnego
- (x_1, x_2, \dots, x_n) – strategia (ciąg decyzji)
- $f_{n-i+1}(y_{i-1})$ – funkcja oceny i-tego etapu i etapów następnych

Metodę PD stosujemy, gdy:

- Stwarzyszona z problemem funkcji celu $Q(x_0, x_1, \dots, x_n)$ ma postać sumy n funkcji (dla każdego z n etapów). Przykładowo:

$$Q = \sum_{i=1}^n g_i(y_{i-1}, x_i) \quad \text{gdzie } g_i \text{ to koszt i-tego etapu}$$

- **Własność Markowa** – decyzja podejmowana w i-tym etapie procesu (oraz stan po etapie) **zależą jedynie od stanu poprzedzającego - y_{i-1}** , a nie od drogi dojścia do tego stanu (decyzji i stanów poprzedzających). To znaczy:

$$Q(x_0, x_1, \dots, x_n) = Q(x_0, x_1, \dots, x_k) + Q(x_k, x_{k+1}, \dots, x_n)$$

Analogiczne z separowaniem funkcjonalów wspominanym wcześniej (str.2)

- **Zasada optymalności Bellmana** (str.1)
- Jeśli mamy zdefiniowane funkcje przejścia między stanami y_{i-1}, y_i po podjęciu decyzji x_i :

$$y_i = T_i(y_{i-1}, x_i) \text{ dla } i = 1, 2, \dots, n$$

T_i jest to **funkcja przejścia** do i-tego etapu.

- Minimalizacja **funkcji celu** ma postać przykładowo:

$$Q = \sum_{i=1}^n g_i(y_{i-1}, x_i) \rightarrow \min$$

Wzór rekurencyjny PD (dla $n \geq 2$) ma postać:

$$f_n(y_0) = \min\{g_1(y_0, x_1)\} + f_{n-1}[T_1(y_0, x_1)]$$

Przykłady zastosowania metody PD:

- Zagadnienie załadunku (problem plecakowy – binarne, całkowitoliczbowe, liniowe, nieliniowe)
- Zagadnienie planowania produkcji
- Zagadnienie optymalizacji zapasów
- Zagadnienie alokacji zasobów (np. siły roboczej, kapitału, maszyn, surowców)
- Zagadnienie optymalnych trajektorii (np. wyboru drogi)
- Zagadnienie planowania inwestycji
- Zagadnienie doboru kontrahentów – dostawców
- Zagadnienie odnowy parku maszynowego
- Zagadnienie planowania kampanii reklamowych

Formalizacja zadania dla PD:

- Dekompozycja procesu decyzyjnego na etapu (wyodrębnienie etapów)
- Definicja zmiennej decyzyjnej (ustalić co jest decyzją)
- Określenie stanu procesu (na podstawie którego podejmujemy decyzje) (coś co się nam zmienia po przejściu kolejnych etapów np. ilość pieniędzy w portfelu)
- Definicja funkcji celu (ocena decyzji – kryterium efektywności całego przedsięwzięcia) (to co nam wyróżnia dlaczego jedna decyzja była lepsza od drugiej np. suma zysków po wszystkich etapach)

- Określenie istotnych ograniczeń dla stanów i decyzji (ograniczenia takie jak np. ograniczona ilość miejsca w magazynie czy limit pieniędzy do wydania)
- Definicja funkcji przejścia między stanami po podjęciu decyzji (ważne z punktu widzenia definicji)
- Struktura danych dla zapisania decyzji optymalnych dla każdego etapu i każdego stanu poprzedzającego (osiągalnego) (ważne z punktu widzenia implementacji).

Metodyka rozwiązywania zadania przy użyciu PD

Rozwiązywanie rozpoczynamy od **ostatniego etapu**:

- Definiujemy funkcję oceny etapu: $f_1(y_{n-1}, x_n)$
- Określamy zbiór stanów dopuszczalnych (osiąganych) przed rozważanym etapem: y_{n-1}
- Dla każdego stanu określamy zbiór decyzji dopuszczalnych: x_n
- Dla wszystkich decyzji dopuszczalnych wyznaczamy funkcję oceny i wybieramy najlepszą
- Zapisujemy dla każdego stanu wejściowego etapu (przed) optymalną decyzję (lub kilka) oraz wartość funkcji oceny (do niej będziemy się rekurencyjnie odwoływać)

Przechodzimy od **przedostatniego etapu** do **pierwszego** postępując jak powyżej:

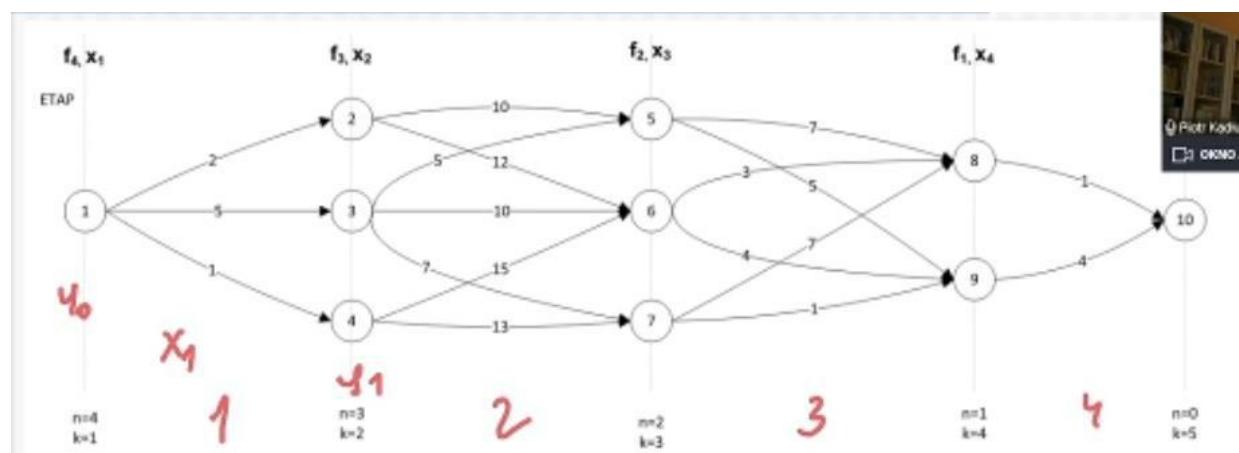
- Definiując rekurencyjnie funkcję oceny aktualnego etapu i etapów następnych: wyrażamy $f_2(y_{n-2}, x_{n-1})$ za pomocą $f_1(y_{n-1}, x_n)$ itd.

W drugiej fazie – mając wyznaczone decyzje optymalne wszystkich etapów i stanów wyznaczamy rozwiązanie (strategię) optymalne całości przedsięwzięcia:

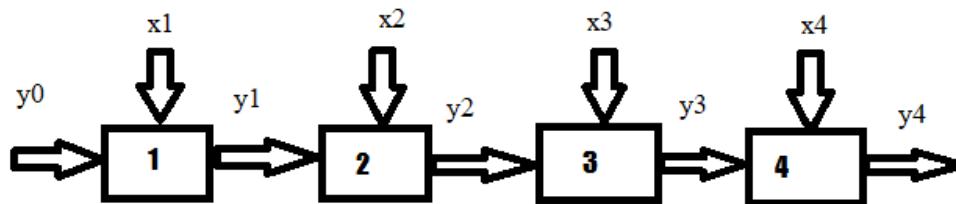
- Dla zadanego **stanu początkowego** odczytujemy optymalną decyzję **etapu pierwszego**
- Wyznaczamy, korzystając z funkcji przejścia stan wyjściowy etapu pierwszego
- Dla tego stanu odczytujemy decyzję w etapie kolejnym, aż do **etapu ostatniego**
- Możemy dodatkowo dokonać sprawdzenia poprawności wyznaczenia funkcji celu dla rozwiązania optymalnego i warunków ograniczających, dotyczących stanów i decyzji

Rozwiązywanie zadania z dyliżansem z wykładu nr.8:

Chcemy się przemieścić z punktu nr.1 do punktu nr.10 co na grafie wygląda następująco:



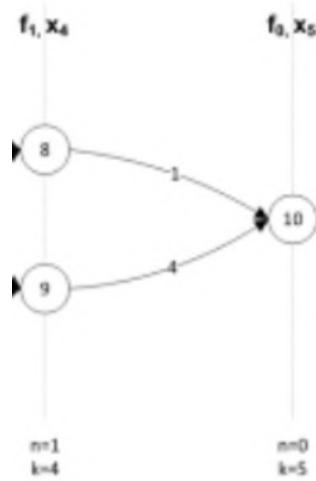
Schematycznie będzie wyglądać:



Dane do zadania:

- Liczba etapów = 4
- Decyzja – wybór celu etapu (nr wierzchołka do którego na danym etapie się udajemy czyli nr wierzchołka na końcu łuku strzałki którą wybieramy). Przykładowo wierzchołek 2 jeśli idziemy z wierzchołka 1
- Stan – miasto w którym aktualnie jesteśmy
- Funkcja celu – suma kosztów połączeń np. 20 dla trasy 1-2-5-8-10
- Ograniczenia:
 - Dla stanów początkowych – przykładowo w etapie nr.2 możemy znajdować się tylko i wyłącznie w wierzchołku 2, 3 lub 4
 - Dla decyzji – przykładowo z wierzchołka nr.2 możemy się udać tylko do wierzchołka 5 lub do wierzchołka 6
- Funkcja przejścia między stanami po podjęciu decyzji – na przykład u nas funkcja przejścia to $y_i = x_i$, czyli w drugim etapie dla stanu 2 podejmując decyzję 5 osiągamy stan 5.
- Struktury danych dla zapisania decyzji optymalnych dla każdego etapu i każdego stanu poprzedzającego – będą to macierze

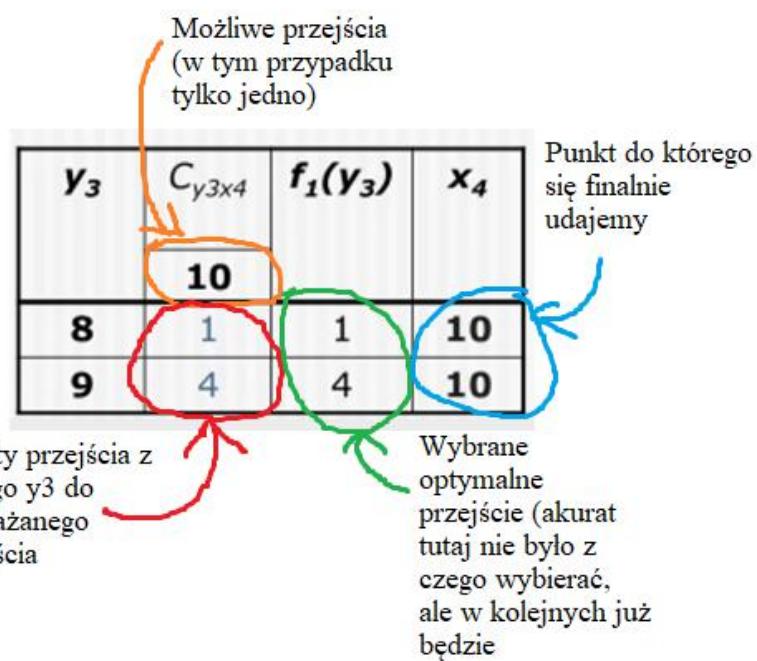
Przechodząc do rozwiązania dla etapu czwartego:



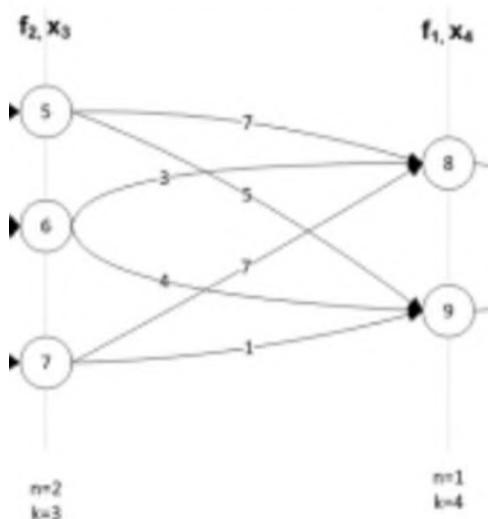
Decyzją będzie zawsze przejście do 10 (bo to nasz punkt końcowy) i wybieramy w stanach wejściowych między 8 i 9 co zapisujemy w tabeli:

y_3	C_{y3x4}	$f_1(y_3)$	x_4
	10		
8	1	1	10
9	4	4	10

Po krótce ją opisując:



Przechodząc do etapu trzeciego:



Tutaj nie będziemy rozważać tylko kosztu przejścia w tym etapie, ale również co się wydarzyło w etapie czwartym, dlatego będziemy korzystać ze wzoru:

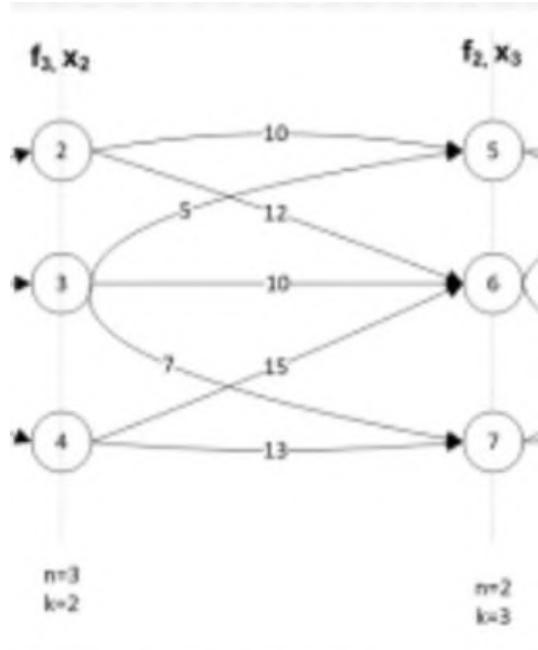
$$f_2(y_2) = \min_{x_3 \in X_3} [C_{y_2 x_3} + f_1(y_3 = x_3)]$$

Tak więc tabela będzie wyglądać następująco:

y_2	$C_{y_2 x_3} + f_1(y_3)$		$f_2(y_2)$	x_3
	8	9		
5	7+1	5+4	8	8
6	3+1	4+4	4	8
7	7+1	1+4	5	9

Jak widać tym razem możliwymi wejściami są 5,6 oraz 7, a możliwe przejścia mamy do 8 lub do 9. Pierwszy człon sumy dla każdego przypadku jest jasny i jest to koszt przejścia z y_2 do potencjalnego x_3 , natomiast drugi składnik jest to wyznaczona z poprzedniej tabeli wartość $f_1(y_3)$ dla danego możliwego x_3 . Jak mówi wzór najmniejsza z dwóch wartości dla danego y_2 trafia do tabelki jako funkcja $f_2(y_2)$.

Dla etapu drugiego:

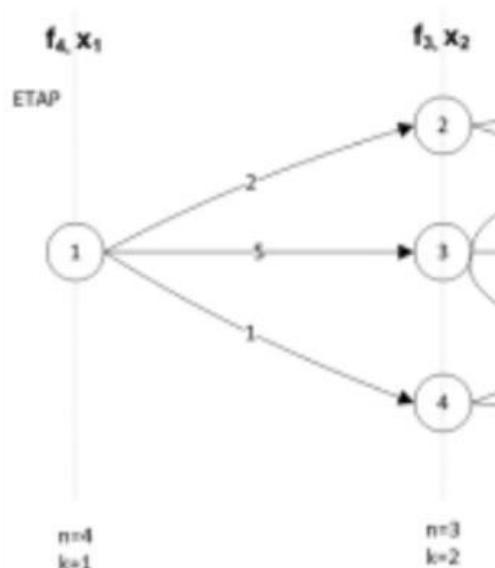


Tutaj możliwymi stanami wyjściowymi są 2, 3 oraz 4, a możliwymi decyzjami są 5,6,7, ale tym razem w porównaniu do poprzednich etapów nie mamy połączeń między każdym z możliwych wierzchołków co będzie widoczne w tabeli.

y_1	$C_{y_1x_2+} + f_2(y_2)$			$f_3(y_1)$	x_2
	5	6	7		
2	10+8	12+4	-	16	6
3	5+8	10+4	7+5	12	7
4	-	15+4	13+5	18	7

W tabelce postępujemy analogicznie jak w etapie poprzednim.

Dochodzimy do etapu pierwszego naszych rozważań:



Jednym możliwym stanem wejściowym jest punkt początkowy naszych rozważań czyli 1, natomiast możliwymi przejściami są 2,3 oraz 4.

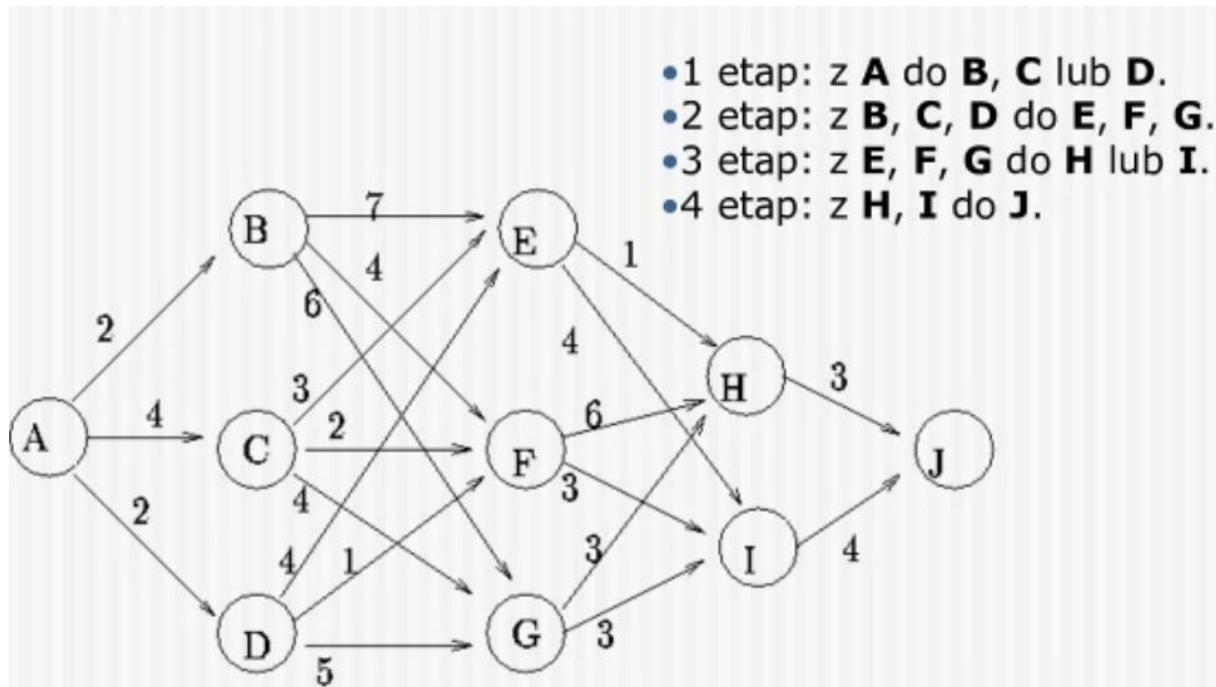
y_0	$C_{y_0x_1+} + f_3(y_1)$			$f_4(y_0)$	x_1
	2	3	4		
1	2+16	5+12	1+18	17	3

Tabelki zawierają zbiór decyzji optymalnych dla każdego etapu i dla każdego stanu i na jej podstawie wyznaczamy ostateczne rozwiązanie:

- Dla etapu pierwszego: stan wejściowy – 1, decyzja – 3
- Dla etapu drugiego: stan wejściowy – 3, decyzja – 7
- Dla etapu trzeciego: stan wejściowy – 7, decyzja – 9
- Dla etapu czwartego: stan wejściowy – 9, decyzja – 10

Rozwiązanie to: 1-3-7-9-10, a koszt takiego rozwiązania to 17.

W przypadku gry trafimy w trakcie poszukiwania minimum na dwie takie same wartości znaczy, że mamy do czynienia z zadaniem, które posiada dwa rozwiązania. Pokażę to drugi przykład:



Rozwiązuje go analogicznie jak przykład poprzedni. Opisywaną powyżej sytuację otrzymuje w etapie drugim:

- Dla etapu **2**
- Stany: **B**, **C**, **D**
- Decyzje: **E**, **F**, **G**

y_1	$C_{y_1x_2+} + f_2(y_2)$			$f_3(y_1)$	x_2
	E	F	G		
B	11	11	12	11	E lub F
C	7	9	10	7	E
D	8	8	11	8	E lub F

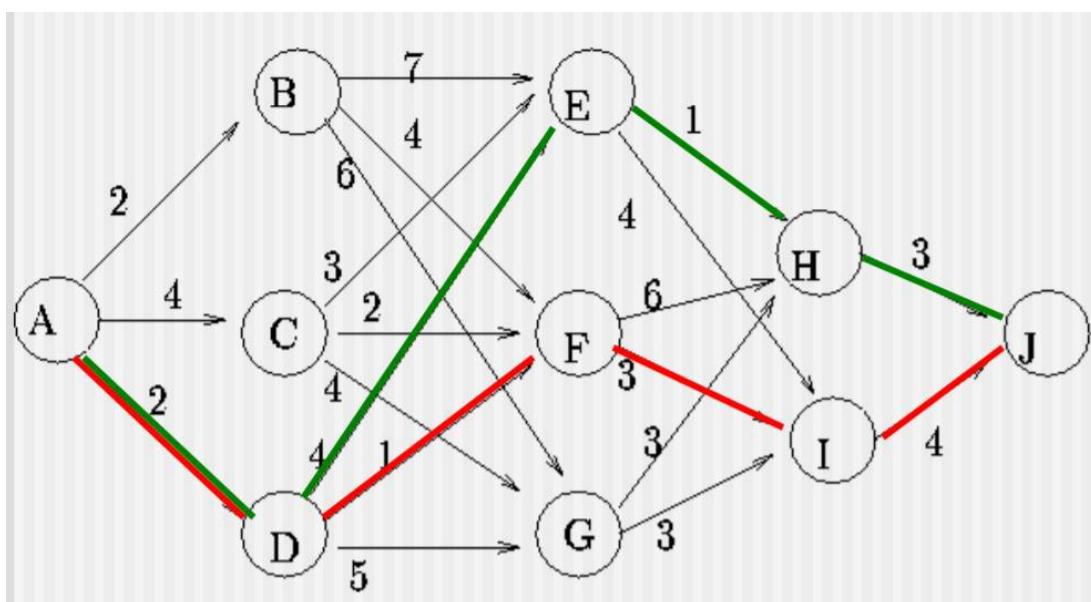
- Dla etapu **1**
- Stany: **A**
- Decyzje: **B**, **C**, **D**

y_0	$C_{y_0x_1+} + f_3(y_1)$			$f_4(y_0)$	x_1
	B	C	D		
A	13	11	10	10	D

Możemy przejść z tym samym kosztem zarówno do wierzchołka E jak i do wierzchołka F, co daje nam 2 możliwe rozwiązania. Tym samym finalne rozwiązania prezentują się następująco:

- Dla etapu 1
 - Stan początkowy **A** – decyzja optymalna **D**
- Dla etapu 2
 - Stan wejściowy **D** – decyzja optymalna **E** lub **F**
- Dla etapu 3
 - Stan wejściowy **E** – decyzja optymalna **H**
 - Stan wejściowy **F** – decyzja optymalna **I**
- Dla etapu 4
 - Stan wejściowy **H** – decyzja optymalna **J**
 - Stan wejściowy **I** – decyzja optymalna **J**
- Optymalne strategie: **ADEHJ**, **ADFIJ**
- Koszt drogi optymalnej: **10**

Co graficznie wygląda następująco:



Problem plecakowy 0-1 KP (0-1 knapsack problem):

Jest to metoda programowania dynamicznego. Mamy plecak o maksymalnej pojemności B oraz zbiór N elementów ($\{x_1, \dots, x_N\}$), gdzie każdy element charakteryzuje się zyskiem c_j i wagą a_j .

Dyskretny problem plecakowy:

Należy maksymalizować $\sum_{j=1}^N c_j x_j$, przy ograniczeniach $\sum_{j=1}^N a_j x_j \leq B$; $x_j \in \{0, 1\}$

Jeśli ujemna jest waga przedmiotu, otrzymamy ujemny zysk, co będzie skutkowało stratą.

Zatem pod uwagę bierzemy tylko przedmioty z dodatnią wagą oraz w tej samej jednostce miary.

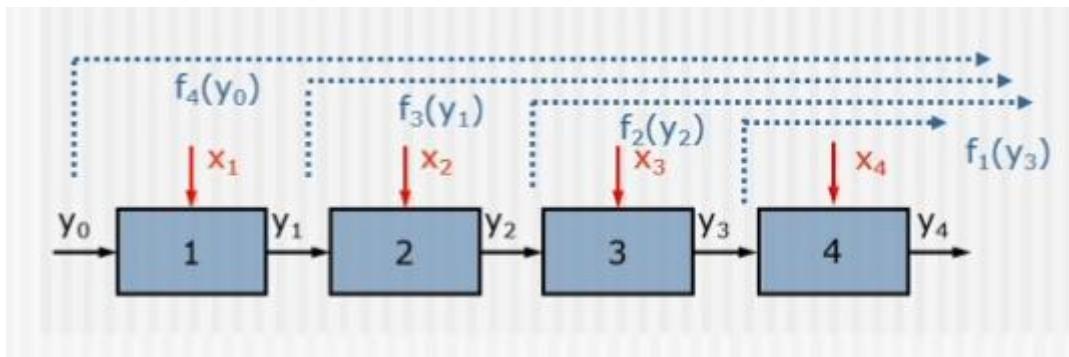
Przykład liczbowy,

gdzie $N = 4$ oraz $B = 7$:

$$c_j: \quad x_1 + 3x_2 + 2x_3 + 2x_4 \rightarrow \max$$

$$a_j: \quad x_1 + 4x_2 + 3x_3 + 3x_4 \leq 7$$

Schemat problemu plecakowego:



Nie jest to proces wieloetapowy!!

Określamy etapy, określamy zmienną decyzyjną, określamy stan procesu (ilość wolnego miejsca w plecaku), przed danym etapem może być określona ilość wykorzystanego miejsca. Funkcja celu sumuje zyski, ograniczenie dla stanów – stany nie mogą spadać poniżej zera. Decyzje – biorę lub nie biorę przedmiot.

Zmiana koncepcji stanu podczas rozwiązywania wiąże się z problemami!

y_{i-1} – ilość wolnego miejsca przed i-tym etapem

$$f_1(y_3) \begin{cases} 0 \text{ dla } x_4 = 0 \text{ jeśli } a_4 > y_3 \\ c_4 \text{ dla } x_4 = 1 \text{ jeśli } a_4 \leq y_3 \end{cases}$$

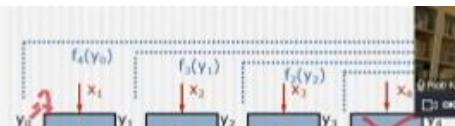
spośród dwóch decyzji biorę tą, która maksymalizuje funkcję.

Krok pierwszy (etap 4, $c_4 = 2$, $a_4 = 3$):

$$f_1(y_3) = \begin{cases} 0 & \text{dla } x_4 = 0 \quad \text{jeśli } a_4 > y_3 \\ c_4 & \text{dla } x_4 = 1 \quad \text{jeśli } a_4 \leq y_3 \end{cases}$$

*0
0, 1
max(0, 2) = 2*

Jeśli a_4 ($3x_4$) jest większe od y_3 , nasza funkcja ($f_1(y_3)$) przyjmie wartość 0 (nie możemy zabrać przedmiotu o wadze większej niż aktualnie możemy wziąć). Jeśli jest mniejsze zabieramy c_4 ($2x_4$):

$x_1 + 3x_2 + 2x_3 + 2x_4 \rightarrow \max$	$x_1 + 4x_2 + 3x_3 + 3x_4 \leq 7$	
$f_1(y_3) = \begin{cases} 0 & \text{dla } x_4 = 0 \quad \text{jeśli } a_4 > y_3 \\ c_4 & \text{dla } x_4 = 1 \quad \text{jeśli } a_4 \leq y_3 \end{cases}$		
y_{i-1}	x_4	$f_1(y_3)$
0	0	0
1	0	0
2	0	0
3	1	2
4	1	2
5	1	2
6	1	2
7	1	2

Krok drugi (etap 3, $c_3 = 2$, $a_3 = 3$):

Jeśli a_3 ($3x_3$) jest większe od y_2 , nasza funkcja ($f_2(y_2)$) przyjmie wartość 0 (analogicznie – nie możemy zabrać przedmiotu o wadze większej niż aktualnie możemy wziąć). Jeśli jest mniejsze wybieramy $f_1(y_2)$ gdy $x_3 = 0$ lub $c_3 + f_1(y_2 - a_3)$ gdy $x_3 = 1$

$$x_1 + 3x_2 + 2x_3 + 2x_4 \rightarrow \max$$

$$x_1 + 4x_2 + 3x_3 + 3x_4 \leq 7$$

$$f_2(y_2) = \begin{cases} f_1(y_2) & \text{dla } x_3 = 0 \\ \max\{f_1(y_2)/c_3 + f_1(y_2 - a_3)\} & \text{dla } x_3 = 1 \end{cases}$$

y_{i-1}	x_4	$f_1(y_3)$	x_3	$f_2(y_2)$	x_2	$f_3(y_1)$	x_1	$f_4(y_0)$
0	0	0	0	0				
1	0	0	0	0				
2	0	0	0	0				
3	1	2	0/1	2				
4	1	2	0/1	2				
5	1	2	0/1	2				
6	1	2	1	4				
7	1	2	1	4				

$f_2(y_2) = f_1(y_3) + f_1(y_2 - a_3)$

$y_3 = y_2 - a_3 \Rightarrow y_3 = 2 - 1 = 1$

$y_2 = 3$

$x_3 = 0 \rightarrow 2 = f_1(2)$

$1 \rightarrow 2 + f_1(3-3) = 2$

$y_2 = 6 \Rightarrow 2$

$x_3 = 0 \rightarrow 2$

$x_3 = 1 \rightarrow 2 + 2 = 4$

Krok trzeci (etap 2 $c_2=3$, $a_2=4$) oraz czwarty (etap 1 $c_1=1$, $a_1=1$):

W następnych krokach postępujemy w ten sam sposób co w poprzednim kroku, zmieniając jedynie analogicznie indeksy wybieranych elementów aż do rozwiązyania problemu:

$$x_1 + 3x_2 + 2x_3 + 2x_4 \rightarrow \max$$

$$x_1 + 4x_2 + 3x_3 + 3x_4 \leq 7$$

$$f_{n-i+1}(y_{i-1}) = \begin{cases} f_{n-i}(y_{i-1}) & \text{dla } x_i = 0 \\ \max\{f_{n-i}(y_{i-1}) / c_i + f_{n-i}(y_{i-1} - a_i)\} & \text{dla } x_i = 1 \end{cases}$$

y_{i-1}	x_4	$f_1(y_3)$	x_3	$f_2(y_2)$	x_2	$f_3(y_1)$	x_1	$f_4(y_0)$
0	0	0	0	0	0	0		
1	0	0	0	0	0	0		
2	0	0	0	0	0	0		
3	1	2	0/1	2	0	2		
4	1	2	0/1	2	1	3		
5	1	2	0/1	2	1	3		
6	1	2	1	4	0	4		
7	1	2	1	4	1	5	0/1	5

$3 + f_2(7-4) = 5$

$y_0 = 7 \rightarrow x_1 = 1$

$y_1 = 7-1=6 \rightarrow x_2 = 1$

$y_2 = 6 \rightarrow x_3 = 1$

$y_3 = 3 \rightarrow x_4 = 1$

$y_4 = 0$

Rozwiązania

$(1, 0, 1, 1) \checkmark$

$(0, 1, 0, 1) \cdot$

$(0, 1, 1, 0) \cdot$

Nieliniowe zagadnienie załadunku:

Zadanie:

Armator statku o **nośności** $b = 7$ [t] ma przewieźć **3 rodzaje** ładunku o wagach:

$$a_1 = 1 \text{ [t/szt]}, a_2 = 2 \text{ [t/szt]}, a_3 = 3 \text{ [t/szt]}$$

W ilości:

$$d_1 = 6 \text{ [szt]}, d_2 = 3 \text{ [szt]}, d_3 = 2 \text{ [szt]}$$

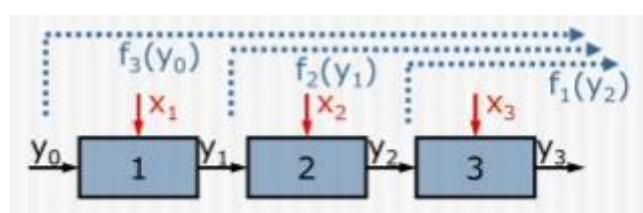
Gdzie:

- Funkcja celu: $q_i(x_i)$
- Decyzja zależy od ilości przedmiotów ($0 \leq x_i \leq d_i$)
- Funkcja jest minimalizowana ($\sum q_i(x_i)$)
- x – ładunek, który przewozimy

W przypadku nie przewiezienia wymaganej liczby jednostek ładunku, armator płaci karę w wysokości $q_i(x_i)$:

x_i	$q_1(x_1)$	$q_2(x_2)$	$q_3(x_3)$
0	20	9	6
1	18	6	2
2	14	3	0
3	11	0	
4	7		
5	2		
6	0		

Wyznacz rozwiązanie minimalizujące sumaryczną karę.

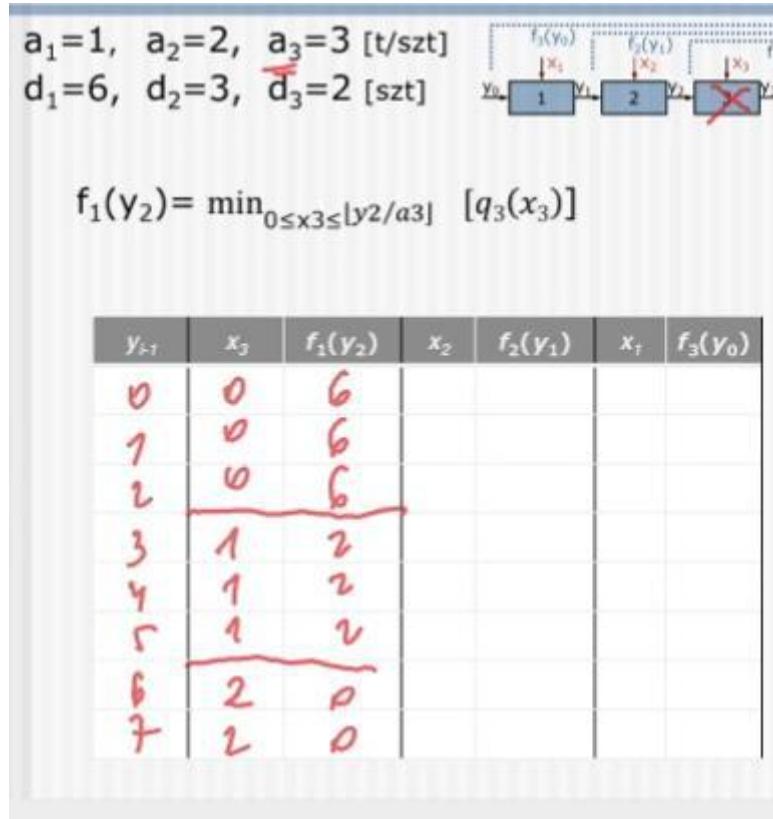


- y_{i-1} – ilość wolnego miejsca przed i-tym etapem
- Funkcja przejścia: $y_i = y_{i-1} - a_i x_i \rightarrow y_3 = y_2 - a_3 x_3$
- Jest co całkowitoliczbowy problem

$$f_1(y_2) = \begin{cases} \min [q_3(x_3)] \\ 0 \leq x_3 \leq \approx \frac{y_2}{a_3} \\ x_3 \leq d_3 \end{cases}$$

Z czego wybieramy rozwiązańe minimalizujące naszą funkcję.

Dla etapu 3 oraz przedmiotu a_3 :



W tym etapie po prostu odczytujemy wartości z tabeli kar

Kolejny etap dla przedmiotu a_2 jest rekurencyjny:

y_{i-1}	x_3	$f_1(y_2)$	x_2	$f_2(y_1)$	x_1	$f_3(y_0)$
0	0	6	0	15		
1	0	6	0	15		
2	0	6	1	12		
3	1	2	0	11		
4	1	2				
5	1	2				
6	2	0				
7	2	0				

$x_2 < d_2$

$f_2(y_1) = \min_{0 \leq x_2 \leq \lfloor y_1/a_2 \rfloor} [q_2(x_2) + f_1(y_1 - a_2 x_2)]$

$y_1 = 2$
 $x_2 = 0 \rightarrow 9 + 6 = 15$
 $1 \rightarrow 6 + 6 = 12$

$y_1 = 3$
 $x_2 = 0 \rightarrow 9 + 2 = 11$
 $1 \rightarrow 6 + 6 = 12$

W dwóch pierwszych przypadkach ($y = 0$ lub 1) nie mogę zabrać dwóch przedmiotów, zatem placę karę dla obu ($9+6=15$). Dla $y = 2$ mogę zabrać jeden przedmiot a_2 oraz zero przedmiotów a_3 , zatem kara wynosi $6+6=12$. Dla $y = 3$, dalej bardziej opłacalne jest zabranie jednego przedmiotu a_3 niż zabieranie przedmiotów a_2 , ponieważ:

$$0 \rightarrow 9 + 2 = 11$$

$$1 \rightarrow 6 + 6 = 12$$

Dla reszty postępuję analogicznie.

Ostatni etap:

W ostatnim etapie (czyli początek pracy) ilość wolnego miejsca wynosi 6 (całkowite wolne miejsce jest równe 7) dla przedmiotu a_1 .

Nieliniowe zagadnienie załadunku

$a_1=1, a_2=2, a_3=3$ [t/szt]
 $d_1=6, d_2=3, d_3=2$ [szt]

$f_3(y_0) = \min_{0 \leq x_1 \leq [y_0/a_1]} [q_1(x_1) + f_2(y_0 - a_1 x_1)]$

$x_1 \leq d_1$

y_{t-r}	x_3	$f_1(y_2)$	x_2	$f_2(y_1)$	x_1	$f_3(y_0)$
0	0	6	0	15		
1	0	6	0	15		
2	0	6	1	12		
3	1	2	0	11		
4	1	2	2	9		
5	1	2	1	8		
6	2	0	3	6		
7	2	0	2	5	5	14

x_t	$q_1(x_1)$	$q_2(x_2)$	$q_3(x_3)$
0	20		
1	18		
2	14		
3	11	0	
4	7		
5	2		
6	0		

$y_0=7$

$x_1 = p \rightarrow 20 + 5$

$1 \rightarrow 18 + 6$

$2 \rightarrow 14 + 8$

$3 - 11 + 5$

$4 - 7 + 11$

$5 - 2 + 12 = 14$

$6 - 0 + 15$

Odczytujemy wartości z tabeli kar i sumujemy z poprzednią wartością funkcji przejścia zgodnie ze wzorem $f_3(y_0)$. Wybieramy najmniejszą wartość i otrzymujemy rozwiązanie dla danego etapu.

Odczytanie rozwiązania polega w tym przypadku na przyjęciu stanu początkowego $y_0 = 7$, dla którego $x_1 = 5$. W związku z tym stan $y_1 = y_0 - x_1 * a_1 \rightarrow 7 - 5 * 1 \rightarrow y_1 = 2$.

Dla $y_1 = 2$, odczytujemy z tabeli $x_1 = 1$. Analogicznie, stan $y_2 = y_1 - x_2 * a_2 \rightarrow 2 - 1 * 2 \rightarrow y_2 = 0$, dla którego $x_3 = 0$. Wektor decyzyjny, który otrzymaliśmy jest równy $(x_1, x_2, x_3) \rightarrow (5, 1, 0)$

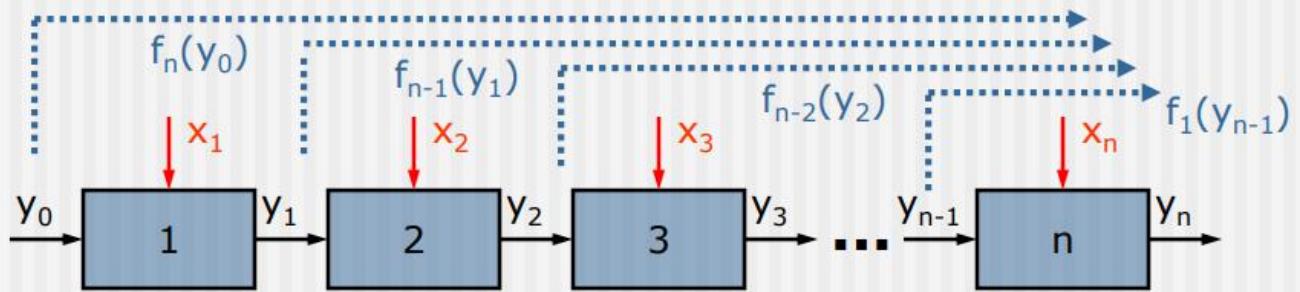
Wykład 10

Programowanie dynamiczne

+ (Wyznaczanie wielkości partii produkcyjnej)

PD:

Schemat wieloetapowego procesu decyzyjnego



- $i = 1, 2, \dots, n$ - etapy procesu decyzyjnego
- n - liczba etapów procesu decyzyjnego
- y_{i-1} - stan wejściowy i -tego etapu (stan przed etapem)
- x_i - decyzja podjęta w i -tym etapie - $x_i \in X_i$ (zbiór decyzji dopuszczalnych etapu)
- y_i - stan wyjściowy i -tego etapu procesu decyzyjnego - $y_i \in Y_i$ (zbiór stanów dopuszczalnych etapu)
- y_0 - stan początkowy procesu decyzyjnego
- y_n - stan końcowy procesu decyzyjnego
- (x_1, x_2, \dots, x_n) - strategia (ciąg decyzji)
- $f_{n-i+1}(y_{i-1})$ - funkcja oceny i -tego etapu i etapów następnych

Formalizacja zadania dla PD

- Dekompozycja procesu decyzyjnego na etapy
- Definicja zmiennej decyzyjnej
- Określenie stanu procesu (na podstawie którego podejmujemy decyzje)
- Definicja funkcji celu (ocena decyzji - kryterium efektywności całego przedsięwzięcia)
- Określenie istotnych ograniczeń dla stanów i decyzji
- Definicja funkcji przejścia między stanami po podjęciu decyzji
- Struktury danych dla zapisania decyzji optymalnych dla każdego etapu i każdego stanu poprzedzającego (osiągalnego)

Najważniejsze dwa slajdy (opisujące etapy wyznaczania rozwiązania):

PD - Metodyka rozwiązywania zadań

Rozwiązywanie rozpoczynamy od **etapu ostatniego**

- Definiujemy funkcję oceny etapu: $f_1(y_{n-1}, x_n)$
- Określamy zbiór stanów dopuszczalnych (osiąganych) przed rozważanym etapem: y_{n-1}
- Dla każdego stanu określamy zbiór decyzji dopuszczalnych: x_n
- Dla wszystkich decyzji dopuszczalnych wyznaczamy funkcję oceny i wybieramy najlepszą
- Zapisujemy dla każdego stanu wejściowego etapu (przed) optymalną decyzję (lub kilka) oraz wartość funkcji oceny

Przechodzimy kolejno od **etapu przedostatniego**, ..., aż do **pierwszego**, powtarzając powyższy schemat

- Definiując rekurencyjnie funkcję oceny aktualnego etapu i etapów następnych: wyrażamy $f_2(y_{n-2}, x_{n-1})$ za pomocą $f_1(y_{n-1})$ itd.

PD - metodyka rozwiązywania zadań - cd

W 2 fazie obliczeń - mając wyznaczone decyzje optymalne dla wszystkich etapów i stanów wyznaczamy rozwiązanie (strategię) optymalne całości przedsięwzięcia:

- Dla zadanego **stanu początkowego** odczytujemy optymalną decyzję **etapu pierwszego**
- Wyznaczamy, korzystając z funkcji przejścia stan wyjściowy etapu pierwszego
- Dla tego stanu odczytujemy decyzję w etapie kolejnym, aż do **etapu ostatniego** – wyznaczając stan końcowy.

- Dodatkowo możemy dokonać sprawdzenia poprawności wyznaczenia funkcji celu dla rozwiązania optymalnego i warunków ograniczających, dotyczących stanów i decyzji.

Zagadnienie wyznaczania wielkości partii produkcyjnej

Zadanie WPP:

Zakład ma zabezpieczyć dostawę produktu w wysokości $q=3$ [szt/mc] przez okres $n=6$ [mc].

Koszt produkcji określa funkcja $g(x)$, koszt magazynowania jednej jednostki w ciągu miesiąca $h=2$ [zł/szt*mc] – jest naliczany wg stanu końcowego po rozpatrywanym miesiącu.

Maksymalna pojemność magazynu wynosi $Y=4$ [szt].

Określić wielkość partii produkcji dla każdego miesiąca przy założeniu, że zapas produktu na początku stycznia i końca czerwca wynosi 0.

X_i [szt]	$g(x_i)$ [zł]
0	0
1	15
2	18
3	19
4	20
5	24

Zadanie WPP

dostawa $q=3$ [szt/mc]
okres $n=6$ [mc].
koszt produkcji - $g(x)$,
koszt magazynowania – $h(=2)*y_i$ [zł]
pojemność magazynu $Y=4$ [szt]

$$y_0 = y_6 = 0$$

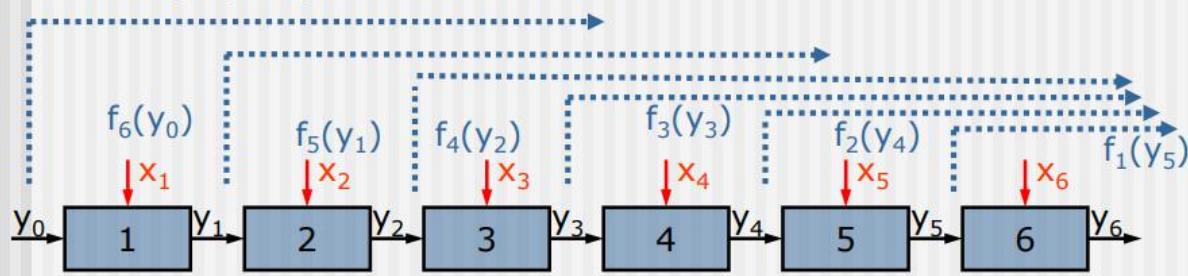
Etap, decyzja

stan

funkcja celu

funkcja przejścia

X_i [szt]	$g(x_i)$ [zł]
0	0
1	15
2	18
3	19
4	20
5	24



Decyzja – liczba wyprodukowanych produktów w rozpatrywanym miesiącu

Stan – liczba sztuk produktu w magazynie przed rozpatrywanym etapem

Funkcja $f_1(y_5)$

Ograniczenia na zbiór decyzji dopuszczalnych:

1. Zdolność produkcyjna zakładu
2. Zapewnienie dostawy q sztuk produktu
3. Maksymalna pojemność magazynu
4. Trafienie w stan końcowy

$$f_1(y_5) = \min_{x_6 \in X_{D6}} [g(x_6) + h * y_6]$$

$$\begin{aligned} f_1(y_5) &= \min [g(x_6) + h * y_6] \\ 0 \leq x_6 &\leq 5 \\ x_6 &\geq q - y_5 \\ x_6 &\leq Y + q - y_5 \\ x_6 &= q - y_5 \end{aligned}$$

$$f_2(y_4) = \min [g(x_5) + h * (y_4 + x_5 - q) + f_1(y_4 + x_5 - q)]$$

$$0 \leq x_5 \leq 5$$

$$x_5 \geq q - y_4$$

$$x_5 \leq Y + q - y_4$$

Krótko mówiąc:

- 1) Najpierw wyznaczamy zbiór możliwych decyzji- na podstawie wszelkich ograniczeń (czyli należy wziąć pod uwagę: ile mamy produktu w magazynie, ile możemy wyprodukować, ile możemy pomieścić w magazynie, ile musimy dostarczyć produktu, w jaki stan końcowy ‘celujemy’, etc)
- 2) Potem dla każdej dopuszczalnej decyzji sprawdzamy jaki koszt się z nią wiąże
- 3) Wybieram decyzję o najmniejszym koszcie i wpisujemy te dane w macierz decyzji

Macierz decyzji dla danych i wzorów podanych powyżej (procedura wyznaczania jest opisana na stronie 2,3):

y_{i-1}	x_6	$f_1(y_5)$	x_5	$f_2(y_4)$	x_4	$f_3(y_3)$	x_3	$f_4(y_2)$	x_2	$f_5(y_1)$	x_1	$f_6(y_0)$
0	3	19	3	38	4	52	3/4	71	3/4	90	4	104
1	2	18	5	30	5	49	5	68	5	82		
2	1	15	4	26	4	45	4/5	64	4	78		
3	0	0	0	19	0	38	0	52	0	71		
4	-	∞	0	20	0	32	0	51	0	70		

Komórki w których są wpisane dwie liczby-> a/b oznaczają, że wyprodukowanie zarówno a jak i b ilości produktu daje taki sam (minimalny względem innych dopuszczalnych decyzji) koszt.

Przykład- Programowanie dynamiczne

Należy ułożyć $L=21[m]$ rurociągu z $n=3$ rodzajów rur, o długościach:

$$l_1=3 \text{ [m/szt]}, l_2=5 \text{ [m/szt]}, l_3=8 \text{ [m/szt]}$$

oraz cenie:

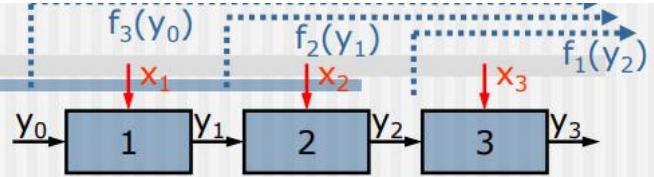
$$k_1=6 \text{ [zł/szt]}, k_2=9 \text{ [zł/szt]}, k_3=14 \text{ [zł/szt]}.$$

W przypadku przekroczenia założonej długości rurociągu płaci się dodatkowo karę $K=1[\text{zł}/\text{m}]$.

Wyznacz rozwiązanie minimalizujące sumaryczny koszt.

PD – Przykład 1

Etap – rodzaj rur n=3



Decyzja – liczba użytych rur danego rodzaju

Stan – brakująca długość rurociągu (do L)

Funkcja celu: $f(X) = \sum_{i=1}^n k_i x_i + K * (\sum_{i=1}^n l_i x_i - L) \rightarrow \min$

Ograniczenia: $\sum_{i=1}^n l_i x_i \geq L$

$$x_i \in Z \geq 0$$

$$f_1(y_2) = \min \{k_3 x_3 + \max[0, K*(l_3 x_3 - y_2)]\}$$

$$f_2(y_1) = \min_{0 \leq x_2 \leq \left\lceil \frac{y_1}{l_2} \right\rceil} \{k_2 x_2 + \max[0, K*(l_2 x_2 - y_1)] + f_1(\max[0, y_1 - l_2 x_2])\}$$

Przykłady programowania dynamicznego

1. Problem układania rurociągu

Należy ułożyć $L=21[m]$ rurociągu z $n=3$ rodzajów rur, o długościach:

$$l_1=3 \text{ [m/szt]}, l_2=5 \text{ [m/szt]}, l_3=8 \text{ [m/szt]}$$

oraz cenie:

$$k_1=6 \text{ [zł/szt]}, k_2=9 \text{ [zł/szt]}, k_3=14 \text{ [zł/szt]}.$$

W przypadku przekroczenia założonej długości rurociągu płaci się dodatkowo karę $K=1[\text{zł}/\text{m}]$.

Wyznacz rozwiązanie minimalizujące sumaryczny koszt.

Formalizacja zadania PD:

- etapem n będzie rodzaj rury: 1, 2, 3
- stanem y będzie ilość metrów brakująca do L . Z treści zadania można wywnioskować, że $y_0 = 21$ (**brakuje całej długości**)
- decyzją x będzie ilość sztuk rury danego typu
- funkcja celu (czyli ta, którą minimalizujemy) wygląda wtedy tak:

$$f(X) = \sum_{i=1}^{n=3} x_i \cdot k_i + K \cdot \left(\sum_{i=1}^{n=3} l_i \cdot x_i - L \right)$$

Narzucamy następujące ograniczenia:

- długość wynikowego rurociągu musi być większa od L :

$$\sum_{i=1}^{n=3} l_i \cdot x_i \geq L$$

- ilość sztuk kupowanych rur jest liczbą naturalną (uwzględniając 0):

$$x_i \in \mathbb{N}$$

- funkcja przejścia (brakuje tyle, ile brakowało – to co doszło):

$$y_i = y_{i-1} - x_i \cdot l_i$$

- funkcja oceny etapów:

$$f_1(y_2) = \min_{\substack{0 \leq x_3 \leq y_2 \\ l_3}} \left(k_3 \cdot x_3 + \max(0, K \cdot (l_3 \cdot x_3 - y_2)) \right)$$

$$f_2(y_1) = \min_{\substack{0 \leq x_2 \leq y_1 \\ l_2}} \left(k_2 \cdot x_2 + \max(0, K \cdot (l_2 \cdot x_2 - y_1)) \right)$$

$$f_3(y_0) = \min_{0 \leq x_1} ((K \cdot l_1 + k_1) \cdot x_1)$$

(tą ostatnią sam wykombinowałem, ale wydaje się być poprawna 😊)

Mając to wszystko już zdefiniowane, możemy przeszukać przestrzeń rozwiązań w dobrze znanej nam tabelce:

- zgodnie z **metodą rozwiązywania** lecimy sobie od tyłu

- w ostatniej kolumnie (pierwsza decyzja) nie wpisujemy niepotrzebnych rozwiązań

y_{i-1}	x_3	$f_2(y_2)$	x_2	$f_3(y_1)$	x_1	$f_4(y_0)$
0	0	0	0	0		
1	1	21	1	13		
2	1	20	1	12		
3	1	19	1	11		
4	1	18	1	10		
5	1	17	1	9		
6	1	16	0	16		
7	1	15	0	15		
8	1	14	0	14		
9	2	35	2	19		
10	2	34	2	18		
11	2	33	1	25		
12	2	32	1	24		
13	2	31	1	23		
14	2	30	3	28		
15	2	29	3	27		
16	2	28	0	28		
17	3	49	2	33		
18	3	48	2	32		
19	3	47	4	37		
20	3	46	4	36		
21	3	45	1	37	0	37

Grafika 1. Przestrzeń rozwiązań brana pod uwagę

Tabelkę tą otrzymuje się rozpatrując przestrzeń decyzji \mathbf{x} i wybierając tą, która minimalizuje funkcję kosztu, zdefiniowaną powyżej (te trzy funkcje f_1, f_2, f_3).

Następnie idąc już zgodnie z kolejnością podejmowanych decyzji (tabela od prawej do lewej, po \mathbf{x}), zbieramy rozwiązanie (pan Piotr już nam to zaznaczył na czerwono).

Ale czemu takie, a nie inne rozwiązanie jest optymalne? – nie jestem pewny, ale mam domysły

Ze zbiorem x_1 nie ma problemu, po prostu bierzemy to, co w nim zostało po odrzuceniu nierealistycznych rozwiązań (w tym kontekście nierealistyczne to takie, które nie spełnia założenia początkowego $y_0 = 21$).

W związku z tym, że decyzja optymalna w zbiorze x_1 to 0, jesteśmy zmuszeni ponownie wybrać opcję dla $y_{i-1} = 21$, bo z funkcji przejścia wynika:

$$y_1 = y_0 - x_1 \cdot l_1$$

a wiemy, że

$$x_1 = 0, y_0 = 21$$

więc

$$y_1 = 21 - 0 \cdot 3 = 21$$

Wybieramy więc opcję $x_2 = 1$.

Żeby wiedzieć, jakie rozwiązania rozpatrywać dalej, ponownie korzystamy z funkcji przejścia:

$$y_2 = y_1 - x_2 \cdot l_2$$

mamy:

$$y_1 = 21, x_2 = 1, l_2 = 5$$

więc:

$$y_2 = 21 - 1 \cdot 5 = 21 - 5 = 16$$

Ostatnia decyzja musi więc spełniać poniższy warunek:

$$x_3 \cdot l_3 \geq 16$$

Ponieważ wtedy osiągniemy wymaganą długość. Okazuje się, że dokładnie to wychodzi dla zaznaczonego na czerwono $x_3 = 2$.

Ostatecznie więc mamy rozwiązanie dokładne:

$$x_1 = 0, x_2 = 1, x_3 = 2$$

czyli

$$(0,1,2)$$

Z kosztem:

$$f(0,1,2) = (6 \cdot 0 + 9 \cdot 1 + 14 \cdot 2) + 1 \cdot (3 \cdot 0 + 5 \cdot 1 + 8 \cdot 2 - 21)$$

Pierwszy nawias to pierwsza suma z funkcji, czyli decyzje i ich koszty, a drugi nawias pozwala wyliczyć karę związaną z nadmiarem długości.

Ostateczna wartość funkcji kosztu wynosi:

$$f(0,1,2) = (9 + 14 \cdot 2) + 1 \cdot (0) = 9 + 28 = 37$$

Rozwiązanie: (0,1,2) koszt: 37 [zł]

2. Reszta przykładów (są one dosyć spoko opisane)

- Alokacja jednostek pieniężnych na naprawy, by zminimalizować awaryjność:

- Dany jest obiekt, będący zespołem urządzeń U₁, U₂, U₃ połączonych szeregowo (awaria jednego z nich jest awarią całego układu - urządzenie ulegają awariom niezależnie).
- Macierz $\mathbf{P} = [\mathbf{p}_{ij}]$, określa prawdopodobieństwa tego, że jeśli na remont *i*-tego urządzenia przeznaczono *j* jednostek pieniężnych, to nie ulegnie ono awarii.

	0 jp	1 jp	2 jp	3 jp	4 jp
U ₁	0.1	0.2	0.4	0.6	0.7
U ₂	0.1	0.3	0.5	0.5	0.8
U ₃	0.3	0.3	0.4	0.6	0.6

- Przedsiębiorstwo przeznaczyło **4** jp na remont całego obiektu.
- Określić optymalny przydział środków, maksymalizujący prawdopodobieństwo braku awarii całego obiektu po dokonaniu remontu poszczególnych urządzeń.

Grafika 2. Slajd z definicją problemu alokacji zasobów

- Kolejny problem alokacji zasobów, wariant gdzie mamy inwestować

Problem alokacji zasobów (decyzja inwestycyjna)

Problem optymalizacji, polegający na takim rozdysponowaniu nakładów inwestycyjnych (6 jp?), aby maksymalizować całkowity zysk inwestycyjny. Wejście metody stanowi macierz zysków zależna od wariantów inwestycyjnych.

Przykładowa macierz zysków:

Nakłady	0	1	2	3	4	5	6
Inwestycja 1	0	6	12	12	12	15	20
Inwestycja 2	0	5	8	11	14	17	18
Inwestycja 3	4	15	15	15	15	15	16

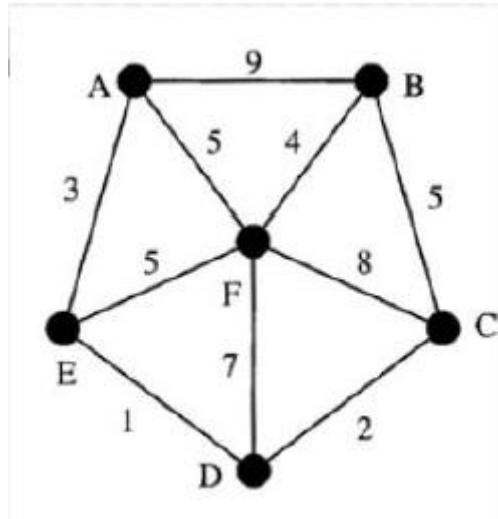
Grafika 3. Problem alokacji z inwestycjami

Metoda podziału i ograniczeń – Branch and bound (B&B)

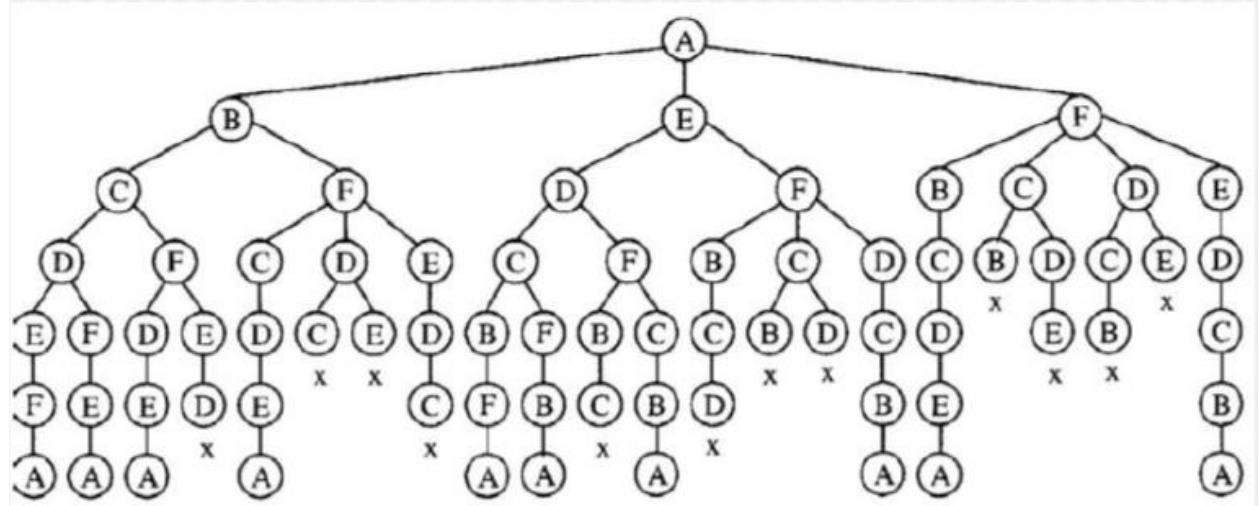
- Jest metodyką ścisłą – znajduje rozwiązanie optymalne
- Stosuje wielokrotny podział na podproblemy, dla których wyznacza granicę (górną lub dolną)
- Przeszukiwanie drzewa podproblemów jest ukierunkowane (np. wg wartości ograniczającej)
- Przypadki zamykania podproblemów:
 - podział problemu
 - znalezienie rozwiązania optymalnego dla podproblemu
 - brak rozwiązań dopuszczalnych
 - gorsza wartość ograniczenia podproblemu od wartości funkcji celu już znalezionego rozwiązania dopuszczalnego

Porównanie drzewa rozwiązań dla zadania testowego TSP:

a) zadanie testowe

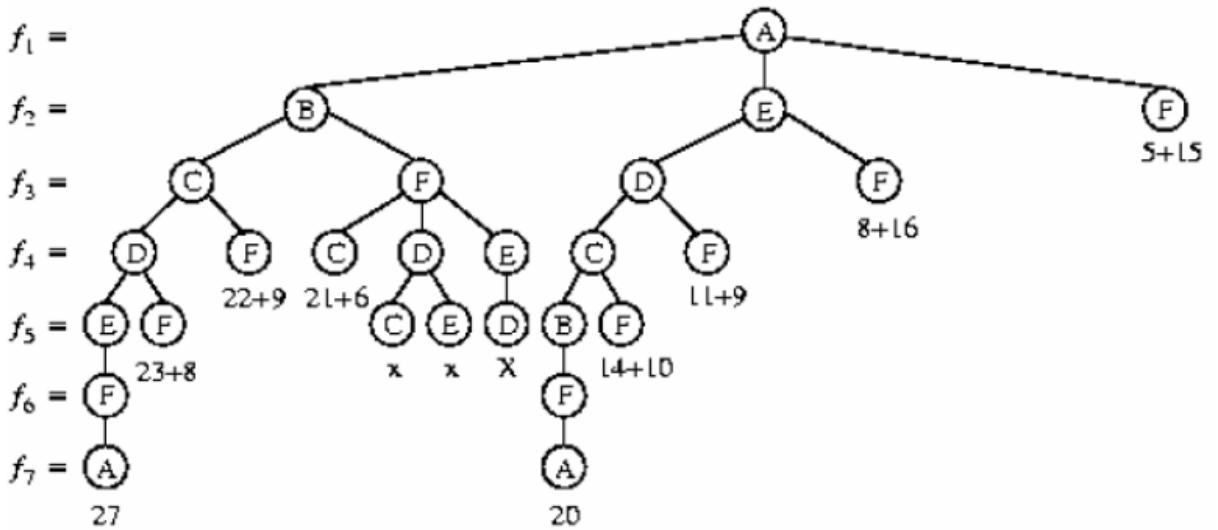


b) drzewo rozwiązań dla przeszukiwania zupełnego



c) drzewo rozwiązań dla metody B&B

- Znalezione rozwiązanie optymalne: **AEDCBFA**
- Wartość funkcji celu: **20**



Wykład 12 i 13 – Metoda podziału i ograniczeń

- Metoda podziału i ograniczeń – ang. Branch and bound (B&B) jest ogólną metodą **dokładną (ścisłą)** – tzn. znajdującą rozwiązanie optymalne – dla różnych problemów optymalizacyjnych (szczególnie zagadnień dyskretnych i kombinatorycznych).
- Stosuje wielokrotny podział przestrzeni rozwiązań na **podproblemy**, dla których wyznaczana jest wartość górnego (UB – dla max) lub dolnego (LB – dla min) ograniczenia.
- Przeszukiwanie drzewa podproblemów jest ukierunkowane (np. wg wartości ograniczającej).
- Podproblemy są **zamykane** gdy:
 - nastąpił podział
 - znaleziono rozwiązanie optymalne PP
 - nie ma rozwiązań dopuszczalnych lub gorszej wartości ograniczenia PP od wartości funkcji celu już znalezionej rozwiązania dopuszczalnego
- Rozwiązanie problemu stanowi **sumę** rozwiązań podproblemów (warunek konieczny podziału)
- **Relaksacja** problemu to opuszczenie najmniej wygodnych ograniczeń (warunków opisujących obszar dopuszczalny)
- **wartość odcinająca** – najlepsza możliwa znana wartość funkcji celu
- **Kryteria zamykania podproblemów:**
 - **KZ1** – jeżeli zbiór rozwiązań zrelaksowanego podproblemu jest pusty
 - **KZ2** – jeżeli wartość funkcji celu jest gorsza niż wartość odcinająca
 - **KZ3** – jeżeli rozwiązanie problemu zrelaksowanego jest dla nas dopuszczalne to jest ono rozwiązaniem problemu. Jeżeli dla niego wartość funkcji cel jest lepsza niż wartość odcinająca, to ją aktualizujemy.

Algorytm B&B

1. Tworzymy listę kandydatów - LK, na której umieszczamy wszystkie niezamknięte problemy
2. Wybór kandydata problemu – KP:
 - jeżeli $LK = \emptyset$ to **STOP**
 - KP – wybór z LK (na podstawie dowolnie wybranej reguły)
 - $LK = LK \setminus \{KP\}$
3. Rozwiąż algorytmem dokładnym problem zrelaksowany RKP
4. Próba zamknięcia KP za pomocą KZ1, KZ2, KZ3
5. Jeśli KP – zamknięty \rightarrow idź do **kroku 2**
6. Podziel KP i jego następcy umieść na LK i przejdź do kroku 2.

Zagadnienie komiwojażera

- znalezienie minimalnego cyklu Hamiltona w grafie ważonym
- mamy macierz odległości
- **macierz zredukowana** – od każdego elementu w wierszu odejmujemy minimum, to samo z kolumnami
- wartość redukcji (czyli suma wszystkich minimów po wierszach i kolumnach) to **dolne oszacowanie** wartości funkcji celu dla wszystkich rozwiązań
- redukcja w wierszu \rightarrow minimalny koszt wyjścia z wierzchołka
- redukcja w kolumnie \rightarrow minimalny koszt wejścia do wierzchołka

Algorytm Little'a

1. Redukcja macierzy i wyznaczenie LB
2. Wyznaczenie odcinek ($i \rightarrow j$) o max optymistycznym koszcie wyłączenia (spośród wszystkich $a_{ij}=0$)
3. Podział problemu na dwa PP:
 - PP1 – zawierający wybrany odcinek ($i \rightarrow j$) – wykreślamy i-ty wiersz oraz j-tą kolumnę, zabraniamy podcyklu i na podstawie dodatkowej redukcji wyznaczamy nowe LB
 - PP2 – niezawierający wybranego odcinka ($i \rightarrow j$) – zabraniamy ($i \rightarrow j$), redukcja, LB
4. Analiza PP i kryterium zakończenia obliczeń:
 - Próba zamknięcia PP za pomocą KZ1, KZ2, KZ3

- Jeżeli wszystkie zamknięte to **STOP**
5. Wybór PP (niezamkniętego) o min wartości LB → idź do kroku 2.

- **Optymistyczny koszt** wyłączenia odcinka wyznaczamy dla wszystkich odcinków zerowych – jest to suma min w wierszu i kolumnie z wyłączeniem tego elementu.
- **Zabronienie przeciwdziałające powstaniu podcyklu** polega na wykluczeniu z drogi odcinka, który wraz z ostatnio włączonym ($i \rightarrow j$) (i wcześniejszymi) domyka podcykl (o długości mniejszej niż n – rozmiar problemu)
 - Znalezienie p – początku łańcucha z odcinkiem ($i \rightarrow j$)
 - Znalezienie k – końca łańcucha z odcinkiem ($i \rightarrow j$)
 - Zabronienie odcinka ($k \rightarrow p$) (tzn. $a_{kp} = \infty$)
- Wielkość redukcji w PP2 po dokonaniu zabronienia ($i \rightarrow j$) (tzn. $a_{ij} = \infty$) jest równa optymistycznemu kosztowi wyłączenia tego odcinka.

Metoda B&B dla TSP (problemu komiwojażera)

I. Redukcja macierzy i wyznaczenie LB (dolnego ograniczenia) -> wiersze i kolumny

Drzewo podproblemów:

- **KZ1** - brak rozw. dop.
- **KZ2** - LB > v^*
- **KZ3** - rozw. dop. = optymalne PP

- II. Odcinek $\langle i^*j^* \rangle$ o maksymalnym optymalnym koszcie wyłączenia
- III. Podział problemu P na podproblemy PP
 - P1: wykreślamy i^* -w. oraz j^* -kol. → zabraniamy podcyklu → redukcja → wyznaczenie LB
 - P2: zabraniamy $\langle i^*j^* \rangle$ → redukcja → LB
- IV. KZ 1-3 i kryterium STOPU
- V. Wybór podproblemu PP(dla minimalnego dolnego ograniczenia LB) -> do **Kroku II**

Zagadnienie komiwojażera - TSP

Dany jest:

- zbiór wierzchołków $\mathbf{N} = \{1, 2, \dots, n\}$
- macierz odległości o rozmiarze $n \times n$ - wymiarowa $\mathbf{A} = [a_{ij}]$

Należy znaleźć rozwiązanie \mathbf{X} , które minimalizuje funkcję celu:

- $\min f(\mathbf{X}) = \sum_{i=1}^m \sum_{j=1}^m a_{ij} x_{ij}$
- ogr. $\sum_{i=1}^n x_{ij} = 1$ dla $j = 1, \dots, n$

$$\sum_{i=1}^n x_{ij} = 1 \text{ dla } i = 1, \dots, n$$

cykl Hamiltona

gdzie $x_{ij} = \{0,1\}$

Zagadnienie przydziału - AP

Dany jest:

- zbiór zadań $\mathbf{N} = \{1, \dots, n\}$
- zbiór maszyn $\mathbf{M} = \{1, \dots, n\}$
- macierz kosztów przydziału ($n \times n$) $\rightarrow \mathbf{A} = [a_{ij}]$ // przydział jeden-jeden zadań i środków

Należy znaleźć rozwiązanie \mathbf{X} , które minimalizuje funkcję celu:

- $\min f(\mathbf{X}) = \sum_{i=1}^m \sum_{j=1}^n a_{ij} x_{ij}$
- ogr. $\sum_{i=1}^n x_{ij} = 1 \text{ dla } j = 1, \dots, n$

$$\sum_{i=1}^n x_{ij} = 1 \text{ dla } i = 1, \dots, n$$

gdzie $x_{ij} = \{0,1\}$

Metoda węgierska AP \rightarrow do rozwiązywania zagadnień przydziału, mamy określone środki i określone zadania, które musimy wzajemnie przydzielić

- I. Redukcja macierzy i wyznaczenie dolnego ograniczenia \mathbf{LB} \rightarrow wiersze i kolumny
- II. Wyznaczenie zbioru zer niezależnych (0^*)
- III. Jeżeli liczba zer niezależnych 0^* wynosi $= n \Rightarrow \mathbf{STOP}$ (rozwiązanie optymalne \mathbf{X} - powiela układ 0^*)
- IV. Próba zwiększenia liczby 0^*
 - Wykreślamy wszystkie 0 macierzy minimalną liczbą linii
 - Wyznaczamy niewykreślony element minimalny el_{min}
 - Odejmujemy el_{min} od niewykreślonych elementów
 - Dodajemy el_{min} do podwójnie wykreślonych
 - Zwiększamy dolne ograniczenie \mathbf{LB} o krotność el_{min}
 - Idziemy do Kroku II.

Metoda węgierska + reguła podziału Bellmore'a – metoda B&B dla TSP:

1. Reguła podziału Bellmore'a:
 - wybieramy podcykle o najmniejszej liczbie wierzchołków
 - zabraniamy przejścia dla wszystkich wierzchołków wybranego podcyklu do pozostałych wierzchołków tego podcyklu
2. Redukujemy macierz po wierszach i kolumnach oraz wyznaczamy LB

3. Wyznaczamy rozwiązania optymalne metodą węgierską dla AP - (relaksacja problemu TSP):
 - Podział P na PP – reguła Bellmore'a
 - Wyznaczenie LB (redukcja)

Ogólne reguły:

- Relaksacja problemu komiwojażera (TSP) do zagadnienia przydziału (AP) przez odrzucenie ograniczenia dotyczącego cyklu Hamiltona.
 - Rozwiązywanie problemu zrelaksowanego algorytmem dokładnym – **metoda węgierska**.
 - Sprawdzeniem dopuszczalności uzyskanego rozwiązania optymalnego AP
 - **Tak** – rozwiązanie optymalne TSP (dla PP)
 - **Nie** – dolne ograniczenie dla rozwiązania optymalnego TSP (LB dla PP)
 - Reguła wyboru podproblemu (KP) – według najmniejszej wartości dolnego ograniczenia
 - Podział PP - Reguła podziału Bellmore'a – narzucenie dopuszczalności przez sprawdzenie wszystkich możliwości

Zagadnienie komiwojażera - typy

1. **Cykl** **Hamiltona**
To taki cykl w grafie, w którym każdy wierzchołek grafu występuje jeden raz. Znalezienie cyklu Hamiltona o minimalnej sumie wag krawędzi jest równoważne rozwiązaniu zagadnienia komiwojażera.
 2. **Grafy** **hamiltonowskie**
Grafy zawierające cykl Hamiltona.
 3. **STSP** Symetryczne zagadnienie komiwojażera
Polega na tym, że odległość z miasta A do B oraz z miasta B do A jest zawsze taka sama.
 4. **ATSP** Asymetryczne zagadnienie komiwojażera
Odległość z miasta A do B może być inna, niż odległość z miasta B do A.
 5. **OTSP** Otwarte zagadnienie komiwojażera
Ścieżka o minimalnej sumarycznej wadze krawędzi (bez powrotu), w której każdy wierzchołek grafu występuje jeden raz.
 6. **Cykl** **Eulera**
To taka zamknięta droga w grafie, która przechodzi przez każdą jego krawędź dokładnie jeden raz.
 7. **Grafy** **eulerowskie**
Grafy zawierające cykl Eulera.

Drzewo rozpinające grafu (ST)

- **Drzewo**
To graf acykliczny i spójny.
 - **G** jest drzewem, jeśli spełnia jeden z warunków:
 - dowolne dwa wierzchołki łączy dokładnie jedna ścieżka prosta

- **G** jest **acykliczny** i dodanie krawędzi łączącej dowolne dwa wierzchołki utworzy cykl
 - **G** jest **spójny** i usunięcie dowolnej krawędzi spowoduje, że G przestanie być spójny
- **Drzewo rozpinające grafu G**
To drzewo, które zawiera wszystkie wierzchołki grafu G, a zbiór krawędzi drzewa jest podzbiorem zbioru krawędzi grafu.
- Konstrukcja **drzewa rozpinającego** polega na usuwaniu z grafu tych krawędzi, które należą do cykli.
- **Rząd acykliczności (liczba cyklowymetryczna)**
Najmniejsza liczba krawędzi jaką trzeba usunąć z grafu, aby graf stał się acykliczny (stał się drzewem).
- **Minimalne drzewo rozpinające (MST)**
To drzewo rozpinające danego grafu ważonego o najmniejszej sumie wag.

- Deterministyczne algorytmy znajdujące dla zadanego grafu **minimalne drzewo rozpinające** (złożoności liniowo-logarytmiczna):
 - Algorytm Boruvki (alg. Sollina)
 - Algorytm Prima (alg. Dijkstry-Prima)
 - Algorytm Kruskala
- Przykład: mamy połączyć siecią łączności n miast, gdzie znane są koszty połączeń między miastami. Poszukiwane rozwiązanie ma minimalizować sumaryczny koszt realizacji sieci.



Rozwiązywanie Algorytmem Dijkstry - Prima

Algorytm Dijkstry - Prima

1. Utwórz drzewo (MST), które początkowo zawiera dowolnie wybrany wierzchołek grafu.
2. Dokonaj cechowania wierzchołków j nienależących do MST (osiągalnych z początkowego wierzchołka - sąsiednich):
 - aktualnie najmniejszy kosztu dotarcia do danego wierzchołka z MST (B_j),
 - nr wierzchołka MST, z którego dołączany jest wierzchołek j (a_j).
3. Powtarzaj, dopóki do MST nie należą wszystkie wierzchołki grafu:
 - wśród wierzchołków nienależących do MST wybierz ten, dla którego koszt dojścia (B_j) jest najmniejszy.
 - dodaj do MST wierzchołek oraz krawędź realizującą najmniejszy koszt
 - zaktualizuj cechowanie wierzchołków, uwzględniając nowe krawędzie wychodzące z dodanego do MST wierzchołka

- **MST** - jest relaksacją problemu OTSP
- **OTSP** – otwarte zagadnienie komiwojażera – ang. open travelling salesman problem
- **MST** – ang. minimum spanning tree
- **SST** – ang. shortest spanning tree