AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

**Problem szeregowania zadań – algorytm Johnsona**

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

**Zad. 1**

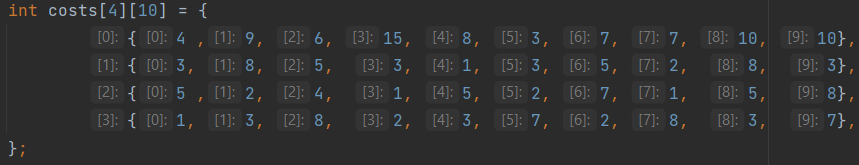
Kod. 1 Zaimplementowany przez mnie algorytmu Campbella-Dudka-Smitha.

#include <iostream>  
#include <list>  
#include <cmath>  
#include <limits>  
#include <tuple>  
  
int inf = std::numeric\_limits<int>::max();  
float float\_inf = std::numeric\_limits<float>::max();  
  
template <size\_t *size*, size\_t *amount*>  
float cal\_cost(int(&costs)[*amount*][*size*], const int \*ans){  
 /\*  
 Funkcja kalkuluje czas skończenia każdego procesu.  
 :param costs: macierz kosztów.  
 :param ans: wskaźnik do tablicy, która zawiera kolejność wykonywania działań.  
 \*/  
  
 // Deklaracja tablicy czasu zakończenia każdego etapu  
 int cal\_cost[*amount*][*size*];  
  
  
 cal\_cost[0][0] = costs[0][ans[0]];  
 for (size\_t y = 1; y != *amount*; y++) {  
 cal\_cost[y][0] = cal\_cost[y - 1][0] + costs[y][ans[0]];  
 }  
  
  
 for (size\_t x = 1; x != *size*; x++) {  
 cal\_cost[0][x] = costs[0][ans[x]] + cal\_cost[0][x - 1];  
  
 for (size\_t y = 1; y != *amount*; y++) {  
 cal\_cost[y][x] = (cal\_cost[y - 1][x] > cal\_cost[y][x - 1]) ? cal\_cost[y - 1][x] : cal\_cost[y][x - 1];  
 cal\_cost[y][x] += costs[y][ans[x]];  
 }  
 }  
  
 return cal\_cost[*amount* - 1][*size* - 1];  
}  
  
  
template <size\_t *size*>  
float johnson(int(&costs)[2][*size*], int \*ans){  
  
 // Deklaracja zmiennych  
 std::list<int> path = {};  
 bool exist[*size*];  
 for(size\_t x = 0; x != *size*; x++) {exist[x] = false;}  
 float cost = 0;  
 int min;  
 std::tuple<int, int> cord = {0, 0};  
 int start = 0;  
 int end = *size* - 1;  
  
 // poszukiwanie kolejności  
 while(end >= start) {  
 min = inf;  
 for (size\_t x = 0; x != *size*; x++) {  
 if (exist[x] == true) {  
 continue;  
 }  
 if (min > costs[0][x]) {  
 min = costs[0][x];  
 cord = {0, x};  
 }  
 if (min > costs[1][x]) {  
 min = costs[1][x];  
 cord = {1, x};  
 }  
 }  
  
 // Wstawianie na początek  
 if (std::get<0>(cord) == 0) {  
 ans[start] = std::get<1>(cord);  
 exist[std::get<1>(cord)] = true;  
 start++;  
 }  
  
 // Wstawianie na koniec  
 if (std::get<0>(cord) == 1) {  
 ans[end] = std::get<1>(cord);  
 exist[std::get<1>(cord)] = true;  
 end--;  
 }  
 }  
  
 return cal\_cost(costs, ans);  
}  
  
template <size\_t *size*, size\_t *amount*>  
float CDS(int(&costs)[*amount*][*size*], int \*ans){  
 // Deklaracja zmiennych  
 int temp\_ans[*size*];  
 int temp\_costs[2][*size*];  
 float min = float\_inf;  
  
 for(size\_t r = 0; r != *amount* - 1; r++){  
 // Tworzenie nowej macierzy kosztów  
 for(size\_t x = 0; x != *size*; x++){temp\_costs[0][x] = 0; temp\_costs[1][x] = 0;}  
  
 for(size\_t x = 0; x != *size*; x++){  
 for(size\_t y = 0; y != r + 1; y++){temp\_costs[0][x] += costs[y][x];}  
 for(size\_t y = *amount* - r - 1; y != *amount*; y++){temp\_costs[1][x] += costs[y][x];}  
 }  
  
 johnson(temp\_costs, temp\_ans);  
 float time = cal\_cost(costs, temp\_ans);  
 if (time < min){  
 min = time;  
 for(size\_t x = 0; x != *size*; x++) {ans[x] = temp\_ans[x];}  
 }  
 }  
  
 return min;  
}  
  
int main() {  
 // Pierwszy przykład  
 int costs1[2][6] = {  
 {9 ,6, 8, 7, 12, 3},  
 {7, 3, 5, 10, 4, 7},  
 };  
  
 int ans1[6] = {0 ,0, 0, 0, 0, 0};  
  
 float cost1 = johnson(costs1, ans1);  
 std::cout << "minimalna dlugosc: "<< cost1 << std::endl;  
  
 for(int an : ans1) {  
 std::cout << an + 1 << " -> ";  
 }  
 std::cout << std::endl << std::endl;  
  
  
 // Drugi przykład  
 int costs2[4][5] = {  
 {12 ,7, 10, 4, 16},  
 {10 ,12, 6, 15, 8},  
 {6 ,18, 8, 13, 6},  
 {15 ,9, 12, 7, 10},  
 };  
  
 int ans2[5] = {0 ,0, 0, 0, 0};  
  
 float cost2 = CDS(costs2, ans2);  
 std::cout << "minimalna dlugosc: "<< cost2 << std::endl;  
  
 for(int an : ans2) {  
 std::cout << an + 1<< " -> ";  
 }  
 std::cout << std::endl << std::endl;  
  
 // Trzeci przykład  
 int costs[4][10] = {  
 {4 ,9, 6, 15, 8, 3, 7, 7, 10, 10},  
 {3, 8, 5, 3, 1, 3, 5, 2, 8, 3},  
 {5 ,2, 4, 1, 5, 2, 7, 1, 5, 8},  
 {1, 3, 8, 2, 3, 7, 2, 8, 3, 7},  
 };  
  
 int ans[10] = {0 ,0, 0, 0, 0, 0, 0, 0, 0, 0};  
  
 float cost = CDS(costs, ans);  
 std::cout << "minimalna dlugosc: "<< cost << std::endl;  
  
 for(int an : ans) {  
 std::cout << an + 1<< " -> ";  
 }  
}

kod źródłowy algorytmu Campbella-Dudka-Smitha.

Tab. 1. Problemu z 10 zadaniami I 4 maszynami.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Z1 | Z2 | Z3 | Z4 | Z5 | Z6 | Z7 | Z8 | Z9 | Z10 |
| M1 | 4 | 9 | 6 | 15 | 8 | 3 | 7 | 7 | 10 | 10 |
| M2 | 3 | 8 | 5 | 3 | 1 | 3 | 5 | 2 | 8 | 3 |
| M3 | 5 | 2 | 4 | 1 | 5 | 2 | 7 | 1 | 5 | 8 |
| M4 | 1 | 3 | 8 | 2 | 3 | 7 | 2 | 8 | 3 | 7 |



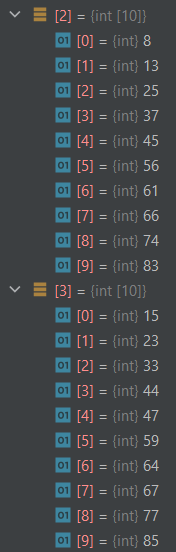
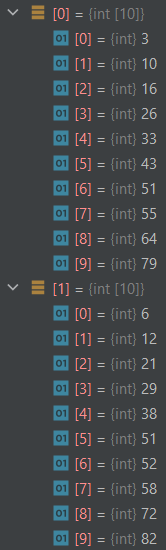
Rys. 1. Reprezentacja problemu.

**Zad. 2**

Początkowo zadania były posegregowane rosnąco.

****

Rys. 2. Kolejność zwrócona przez program.

****

Rys. 3. Czasy wykonania poszczególnych zadań.

Na zielono będą zaznaczone wartości zweryfikowane z programem.

Tab. 2. Macierz dla r = 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r = 1 | Z1 | Z2 | Z3 | Z4 | Z5 | Z6 | Z7 | Z8 | Z9 | Z10 |
| M1' | 4 | 9 | 6 | 15 | 8 | 3 | 7 | 7 | 10 | 10 |
| M2' | 1 | 3 | 8 | 2 | 3 | 7 | 2 | 8 | 3 | 7 |
| Kolejność 6, 3, 8, 10, (2, 9, 5), (4, 7), 1  Program 6, 3, 8, 10, 9, 5, 2 , 7, 4, 1  Tab. 3. Macierz dla r = 2 | | | | | | | | | | |
| r = 2 | Z1 | Z2 | Z3 | Z4 | Z5 | Z6 | Z7 | Z8 | Z9 | Z10 |
| M1' | 7 | 17 | 11 | 18 | 9 | 6 | 12 | 9 | 18 | 13 |
| M2' | 6 | 5 | 12 | 3 | 8 | 9 | 9 | 9 | 8 | 15 |
| Kolejność 6, 3, 10, (7, 8), (9,5), 1, 2, 4 lub 6, 8, 3, 10, 7, (9,5), 1, 2, 4  Program 6, 8, 3, 10, 7, 9, 5, 1, 2, 4  Tab. 4. Macierz dla r = 3 | | | | | | | | | | |
| r = 3 | Z1 | Z2 | Z3 | Z4 | Z5 | Z6 | Z7 | Z8 | Z9 | Z10 |
| M1' | 12 | 19 | 15 | 19 | 14 | 8 | 19 | 10 | 23 | 21 |
| M2' | 9 | 13 | 17 | 6 | 9 | 12 | 14 | 11 | 16 | 18 |

Kolejność 6, 8, 3, 10, 9, 7, 2, (5, 1) , 4

Program 6, 8, 3, 10, 9, 7, 2, 5, 1, 4

Wszystkie kolejności wykonywania zrobiły się poprawnie

Tab. 5. Tabela czasów zakończenia dla r = 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r = 1 | Z6 | Z3 | Z8 | Z10 | Z9 | Z5 | Z2 | Z7 | Z4 | Z1 |
| M1 | 3 | 9 | 16 | 26 | 36 | 44 | 53 | 60 | 75 | 79 |
| M2 | 6 | 11 | 18 | 29 | 37 | 45 | 61 | 66 | 78 | 82 |
| M3 | 8 | 15 | 19 | 27 | 42 | 50 | 63 | 73 | 79 | 87 |
| M4 | 15 | 23 | 31 | 38 | 45 | 53 | 66 | 75 | 81 | 88 |

Tab. 5. Tabela czasów zakończenia dla r = 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r = 2 | Z6 | Z8 | Z3 | Z10 | Z7 | Z9 | Z5 | Z1 | Z2 | Z4 |
| M1 | 3 | 10 | 16 | 26 | 33 | 43 | 51 | 55 | 64 | 79 |
| M2 | 6 | 12 | 21 | 29 | 38 | 51 | 52 | 58 | 72 | 82 |
| M3 | 8 | 13 | 25 | 33 | 45 | 56 | 61 | 66 | 74 | 83 |
| M4 | 15 | 23 | 33 | 40 | 47 | 59 | 64 | 67 | 77 | 85 |

Tab. 5. Tabela czasów zakończenia dla r = 3

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r = 3 | Z6 | Z8 | Z3 | Z10 | Z9 | Z7 | Z2 | Z5 | Z1 | Z4 |
| M1 | 3 | 10 | 16 | 26 | 36 | 43 | 52 | 60 | 64 | 79 |
| M2 | 6 | 12 | 21 | 29 | 44 | 49 | 60 | 61 | 67 | 82 |
| M3 | 8 | 13 | 25 | 37 | 49 | 56 | 62 | 67 | 72 | 83 |
| M4 | 15 | 23 | 33 | 44 | 52 | 58 | 65 | 70 | 73 | 85 |

**Zad. 3.**

**Jaki typ problemu rozwiązujemy (klasyfikacja Grahama)?**

Rozwiązujemy problemy typu flow shop bo wszystkie zadania są wykonywane według konkretnego przepisu na kolejnych maszynach.

**Jakie czasy uzyskamy przy alternatywnych sposobach uszeregowania (taki samo min)?**

Podczas korzystania z algorytmu Johanssona dla 2 lub 3 maszyn wynik będzie taki sam. W przypadku CDS otrzymujemy rozwiązanie przybliżone więc wynik może się różnić.

**Jakie warunki są konieczne w realizacji algorytmu / co jeśli nie będzie spełniony?**

* Stały czas na wykonanie każdego zadania

Algorytm się nie nadaje do rozwiązywania takiego zadania

* Brak równoległości pracy

Algorytm się nie nadaje do rozwiązywania takiego zadania

* Brak priorytetów

Algorytm się nie nadaje do rozwiązywania takiego zadania

**Jaka jest złożoność obliczeniowa algorytmu?**Moja implementacja ma złożoność:

**Wnioski**

Problem przydziału maszyn jest ważnym oraz trudnym problemem. Algorytm CDS jest prosty do napisania ale nie jest idealny. Ćwiczenie okazało się przyjemne. Wiem, że mój algorytm nie jest napisany idealnie ale mimo wszystko jestem zadowolony, że udało mi się go napisać.