AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Drzewa wszystkich najkrótszych ścieżek - algorytm Dijkstry

Aplikacja równoległa MPI

Arkadiusz Kasprzak, Aleksandra Poręba

Agenda

1. Algorytm Dijkstry - wprowadzenie
2. Budowa i działanie projektu
3. Przeprowadzone testy
4. Kompilacja i uruchomienie

Algorytm Dijkstry - wprowadzenie

- Cel: znalezienie najkrótszych ścieżek z wybranego wierzchołka grafu do wszystkich pozostałych wierzchołków.
- Operujemy na grafie skierowanym lub nieskierowanym o nieujemnych wagach krawędzi.
- Algorytm zachłanny - w każdym kroku algorytmu wybierany jest wierzchołek o najmniejszej wartości kosztu.
- Możliwość znalezienia zarówno najkrótszych ścieżek, jak i ich kosztów.

Działanie projektu - wprowadzenie

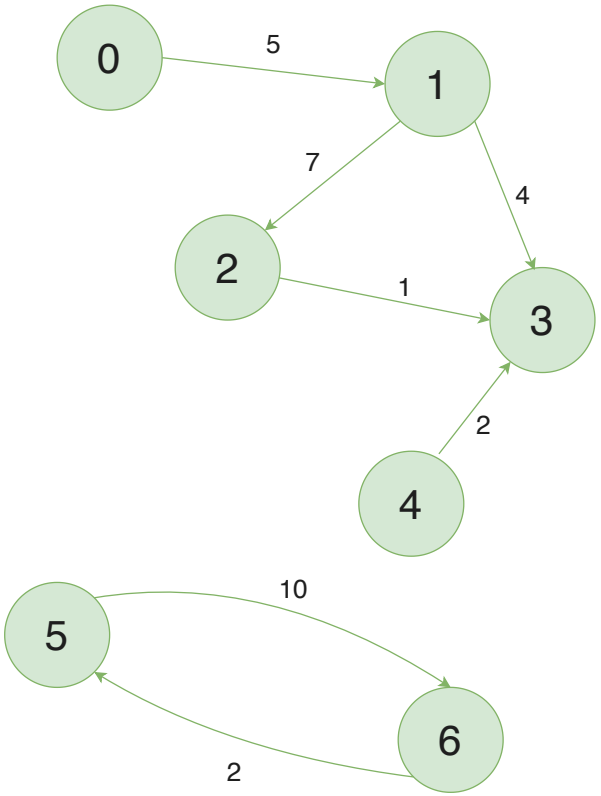
Program podzielony został na 3 części:

- 1 - inicjalizacja: wczytanie, walidacja i podział danych wejściowych, przygotowanie infrastruktury MPI, wykluczenie z dalszych części nadmiarowych procesów
- 2 - wykonanie algorytmu przez wyznaczone do tego celu procesy
- 3 - zapis wyników do pliku, zwolnienie zasobów MPI

Szczegółowy opis każdej z części wraz ze schematami blokowymi zawarty został w dokumentacji projektu.

Działanie projektu - dane wejściowe

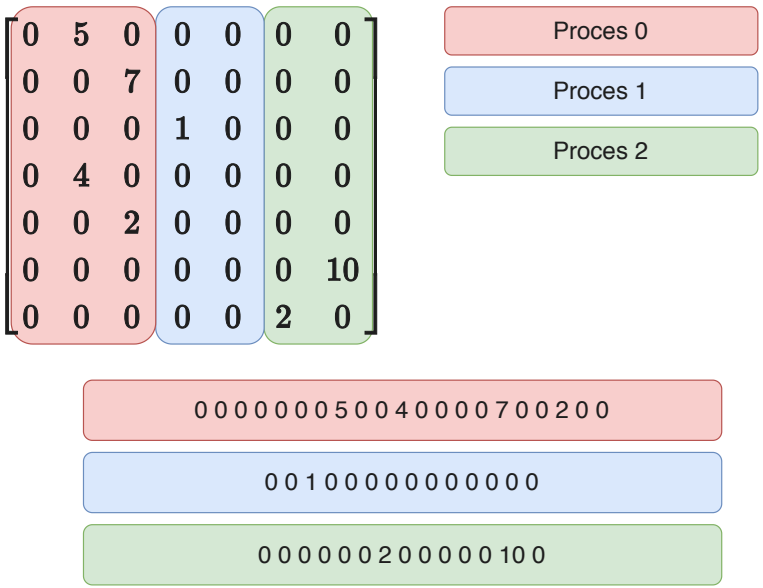
- Dane podawane w postaci tzw. macierzy sąsiedztwa.
- Dane wczytywane z pliku przez proces o numerze 0.



$$\begin{bmatrix} 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

Działanie projektu - podział danych wejściowych

- Dane dzielone są kolumnami równomiernie pomiędzy wszystkie procesy.
- Jeśli podział równomierny niemożliwy, pozostałe k kolumn dzielone jest między pierwsze k procesów.
- Procesy otrzymują następujące po sobie kolumny w formie tablic 1D.



Działanie projektu - podział danych wejściowych

- Jeśli liczba kolumn mniejsza niż liczba procesów, procesy nadmiarowe wykluczane z dalszego wykonywania algorytmu (poprzez podział komunikatora globalnego na dwie części)
- Wykorzystane funkcjonalności MPI: `MPI_Bcast`, `MPI_Scatter`, `MPI_Scatterv`, `MPI_Comm_split`

Działanie projektu - przebieg algorytmu

- Operujemy na dwóch tablicach: d - tablica kosztów, p - tablica poprzedników
- Każdy proces przechowuje swoją część tablicy, zgodnie z podziałem macierzy sąsiedztwa.
- Tablica kosztów inicjalizowana nieskończonościami, tablica poprzedników - wartościami -1.
- Koszt dla wierzchołka źródłowego ustawiany na 0.
- Dodatkowo przechowujemy zbiór wierzchołków przetworzonych przez algorytm - początkowo jest on pusty.

Rys.: Przykładowy początkowy stan tablic

Działanie projektu - przebieg algorytmu

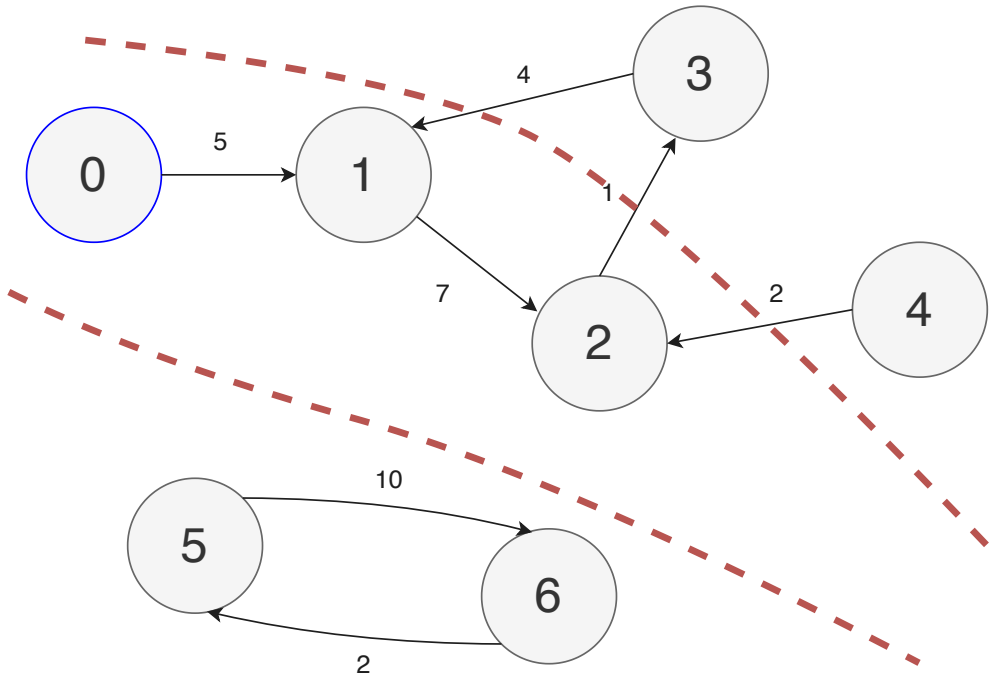
W dalszej części powtarzamy poniższe czynności aż do momentu, gdy wspomniany zbiór zawierał będzie wszystkie wierzchołki:

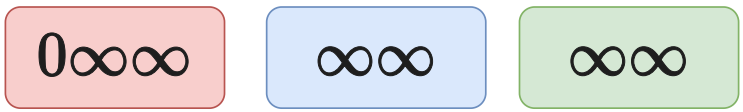
- każdy proces spośród przydzielonych mu nieprzetworzonych jeszcze wierzchołków wybiera ten o najmniejszej wartości kosztu
- spośród wybranych wierzchołków wyznaczany jest ten o globalnie najniższym koszcie (operacja `MPI_Allreduce`). Zostaje on dodany do zbioru wierzchołków przetworzonych.
- na podstawie wylosowanego wierzchołka przeprowadzana jest aktualizacja w tabelach kosztów i poprzedników

Po zakończeniu działania algorytmu fragmenty obu tablic są łączone za pomocą operacji `MPI_Gatherv`. Wyniki algorytmu zapisywane są przez proces 0 do pliku.

Działanie projektu - przebieg pojedynczej pętli

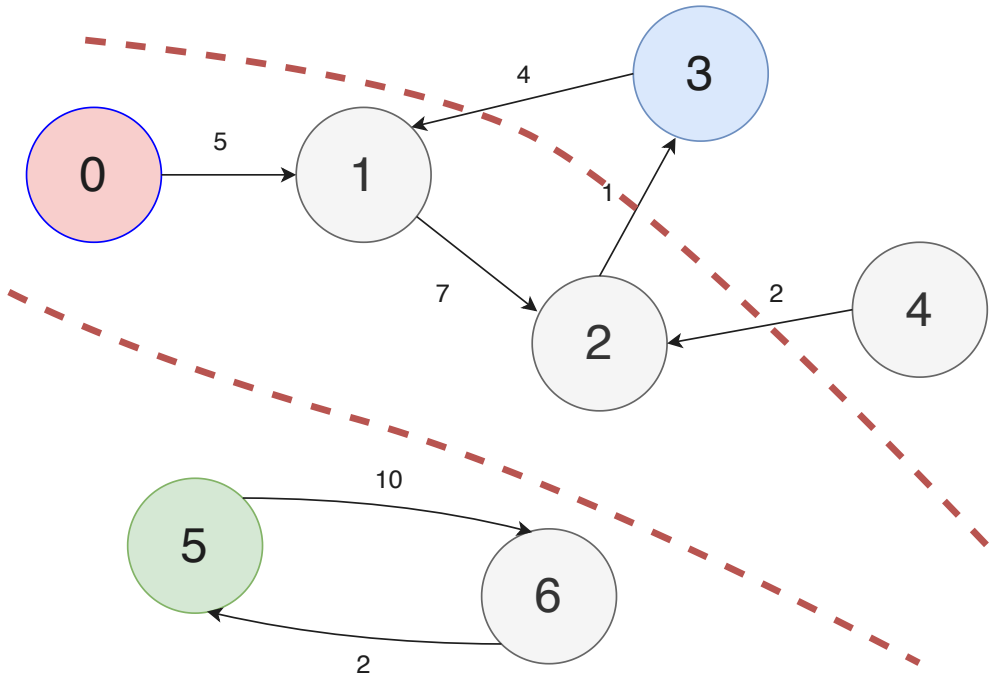
Sytuacja początkowa. Niebieski - wierzchołek źródłowy. Czerwone linia - podział grafu. Na dole koszty.

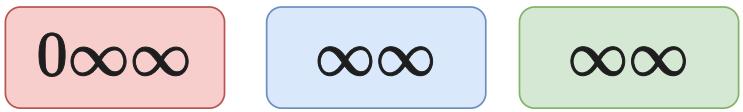




Działanie projektu - przebieg pojedynczej pętli

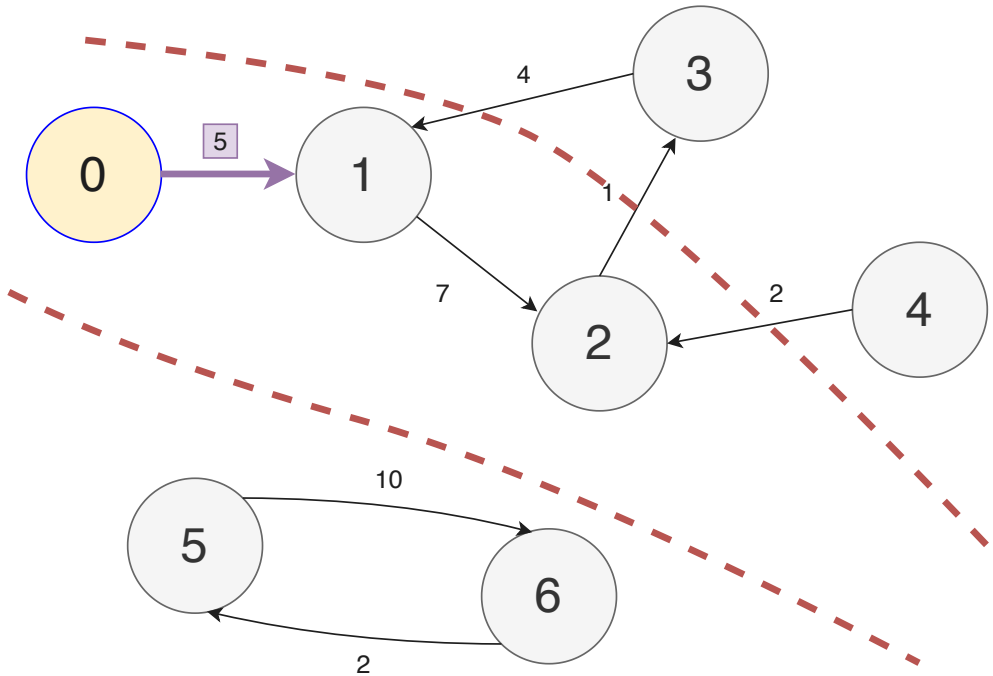
Wybór nieprzetworzonych wierzchołków o najniższym koszcie (w tablicy) w każdym z procesów.





Działanie projektu - przebieg pojedynczej pętli

Wybór wierzchołka o globalnie najniższym koszcie (`MPI_Allreduce`) i dodanie go do zbioru. Aktualizacja kosztów i poprzedników.



$0, 5, \infty$ ∞, ∞ ∞, ∞

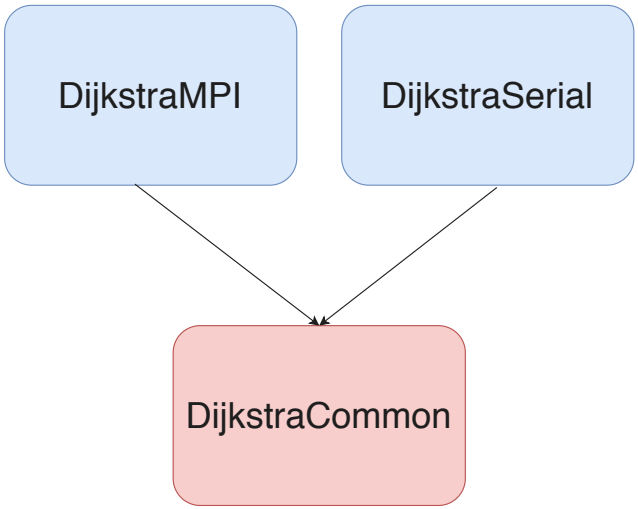
Interfejs

- Podział grafu pomiędzy poszczególne procesy jest przedstawiany użytkownikowi.
- Podobnie przedstawiany jest czas wykonywania się każdego z etapów programu (z punktu widzenia procesu 0).

```
[Process 1] This process will handle 2 vertices in range [3, 4]
[Process 2] This process will handle 2 vertices in range [5, 6]
[Process 0] This process will handle 3 vertices in range [0, 2]
[Process 0] Total elapsed time: 0.0107225s
[Process 0] Setup took: 0.00937863s
[Process 0] Algorithm took: 0.000550448s
[Process 0] Printing solution took: 0.000793405s
```

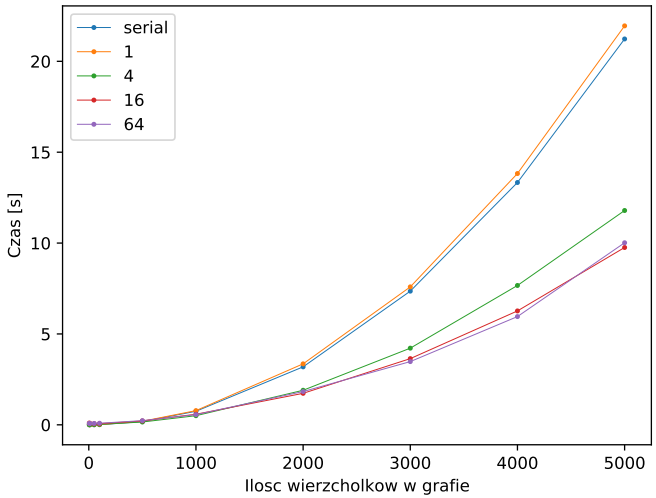
Komponenty projektu

- `DijkstraMPI` - implementacja równoległa algorytmu Dijkstry
- `DijkstraSerial` - implementacja sekwencyjna algorytmu Dijkstry
- `DijkstraCommon` - wspólna część obu implementacji, biblioteka statyczna

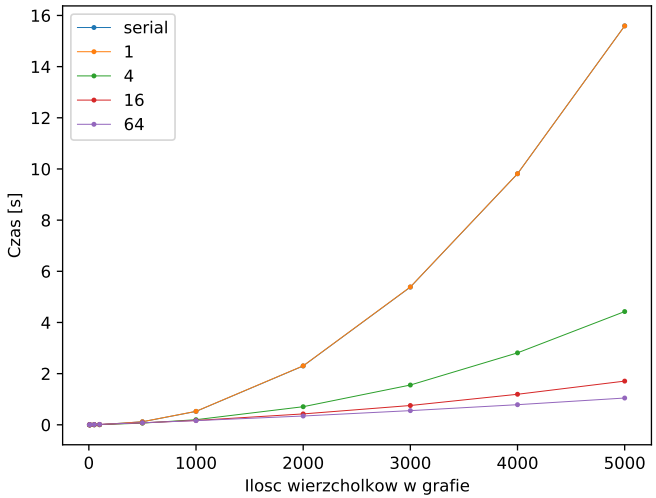


Testy wydajności

Przeprowadzone zostały testy wydajności porównujące wersję sekwencyjną i równoległą projektu. W przypadku wersji równoległej przetestowano użycie 1, 4, 16 i 64 procesów.



(a) Cały program



(b) Część algorytmiczna

Kompilacja i uruchomienie

- Przejście do katalogu `build_uni`
- Przygotowanie środowiska pracowni
- Wykonanie polecenia `make`
- Wykonanie polecenia `make runMPI` z opcjonalnymi argumentami określającymi: plik z węzłami, ilość procesów, numer wierzchołka źródłowego oraz ścieżkę do pliku z danymi
- Wynik dostępny w pliku `resultsMPI.txt`
- Proces szczegółowo opisany w dokumentacji