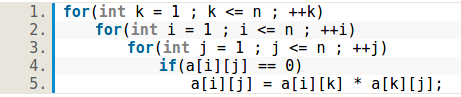
**Roy-Floyd**

Algoritmul Floyd-Warshall(intalnit si sub numele de Roy-Floyd) este un algoritm pentru găsirea celor mai scurte căi într-un grafic ponderat direcționat cu greutăți de margine pozitive sau negativ, poate fi utilizat si in grafuri care contin cicluri de costuri negative. O singură execuție a algoritmului va găsi lungimile celor mai scurte căi între toate perechile de vârfuri.

Complexitatea temporala este O(), iar cea spațială este tot O(). O complexitate spațială cu un ordin mai mic se obține observând ca la un pas nu este nevoie decât de matricea de la pasul precedent d[i, j, k-1] si cea de la pasul curent d[i, j, k]. O observație și mai bună este că, de la un pas k-1 la k, estimările lungimilor nu pot decât sa scadă, deci putem sa lucram pe o singura matrice. Deci, spațiul de memorie necesar este de dimensiune .

Implementarea secventiala:



Am implementat secvential acest algoritm in 3 limbaje:

* C++
* C# (console application cu .net 6)
* Python

Pe axa absciselor sunt reprezentate perechea de valori n\_m unde n este numarul de noduri si m numarul de muchii. Din diagrama este evident faptul zis anterior ca timpul de executie a algoritmului Roy-Floyd nu depinde de numarul de muchii.

In concluzie putem observa ca din toate implementarile prezentate mai sus a algoritmului Roy-Floyd, cea mai rapida este implementarea in limbajul C++ cu Array, deci timpii de executie a aceste implementari vor fi folositi pentru a urmari eficienta algoritmilor paraleli pentru Roy-Floyd.   
Mai jos putem vedea acesti timpi.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 20 | 30 | 40 | 50 | 100 | 200 | 300 | 400 | 500 |
| T(µs) | 4 | 38 | 125 | 285 | 548 | 15K | 168K | 146K | 297K | 607K |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1k | 1.5k | 2k | 2.5k | 3k | 5k | 7k | 10k |
| T(s) | 4 | 19 | 38 | 72 | 118 | 495 | 1295 | 4286 |

**Timpii de executie pentru Parallel Roy-Floyd**

Timpii sunt masurati in µs.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 20 | 30 | 40 | 50 | 100 | 200 | 300 | 400 | 500 |
| 2 Pr | 54 | 76 | 132 | 254 | 495 | 5K | 29K | 102K | 697K | 714K |
| 5 Pr | 869 | 4K | 14K | 16K | 26K | 44K | 68K | 128k | 217K | 320K |
| 100 Pr |  |  |  |  |  | 7M | 7M | 6M | 28M | 25M |

Timpii sunt musurati in Ms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1k | 1.5k | 2k | 2.5k | 3k | 5k | 7k | 10k |
| 2 Pr | 3K | 8K | 16K | 38K | 69K | 304K | 801K | 2465K |
| 5 Pr | 2311 | 10K | 21K | 42K | 66K | 279K | 696K | 1957K |
| 100Pr | 47K | 419K | 567K | 760K | 846K | 1395K |  |  |

In concluzie putem zice ca cea mai rapida implementare paralela este cea cu MPI cu 5 procese.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 20 | 30 | 40 | 50 | 100 | 200 | 300 | 400 | 500 |
| T(µs)p | 17 | 50 | 127 | 242 | 461 | 5K | 39K | 99K | 658K | 586K |
| T(µs) s | 4 | 38 | 125 | 285 | 548 | 15K | 168K | 146K | 297K | 607K |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1k | 1.5k | 2k | 2.5k | 3k | 5k | 7k | 10k |
| T(s)p | 3 | 8 | 21 | 39 | 67 | 305 | 696 | 2095 |
| T(s)s | 4 | 19 | 38 | 72 | 118 | 495 | 1295 | 4286 |

Observam ca pentru seturi mari de date obtine o accelerarea= 2, deci putem zice ca algoritmul nostru plus implementarea contribuiesc la o accelerare buna.

Concluzii:

* Pentru a implementa algoritmul Roy- Floyd, limbajul Python este total nepotrivit, fiind un limbaj de nivel inalt
  + Limbajul Python este un limbaj foarte dinamic din acest motiv compilatorul python are nevoie sa execute mai operatii pentru a determina mereu cum se va comporta in anumite circumstante. Pe cand in limbajul C++ comportamentul obiectelor este predefinit inainte de rulare.
  + <https://stackoverflow.com/questions/16899332/why-does-trivial-loop-in-python-run-so-much-slower-than-the-same-in-c-and-how>
* Conteaza foarte mult structurile de date folosite pentru stocarea matricei de adiacenta.  
  Au fost deja prezentate unele situatii in care o structura de date este mai eficienta( ne ofera timpi mici la executarea programului).
* Pentru implementarea paralela, daca nu despunem de un calculator „adecvat”, cea mai eficienta metoda de a paraleliza alg. Roy-Floyd este utilizarea STL-ului, care necesita doar 2 lini de cod extra si nu necesita teste speciale pe granularitatea task-urilor, datelor etc..
* Dintre cele 4 implementari paralele, cel mai eficient(pe calculatorul personal) a fost implementarea cu MPI (C++) rulata pe 5 procese, urmata de implementarea STL (C++).  
  Avand in vedere ca nucleul sistemelor de operare(Windows, MacOS, Linux) a fost scris in mare parte in limbajul C. La Windows avand si majoritate sistemele si subsistemelor ale SO scrise in limbaju C++, avem la dispozitie optimizari ale compilatorului si limbajului care ne dau o accelerare la executia programelor.