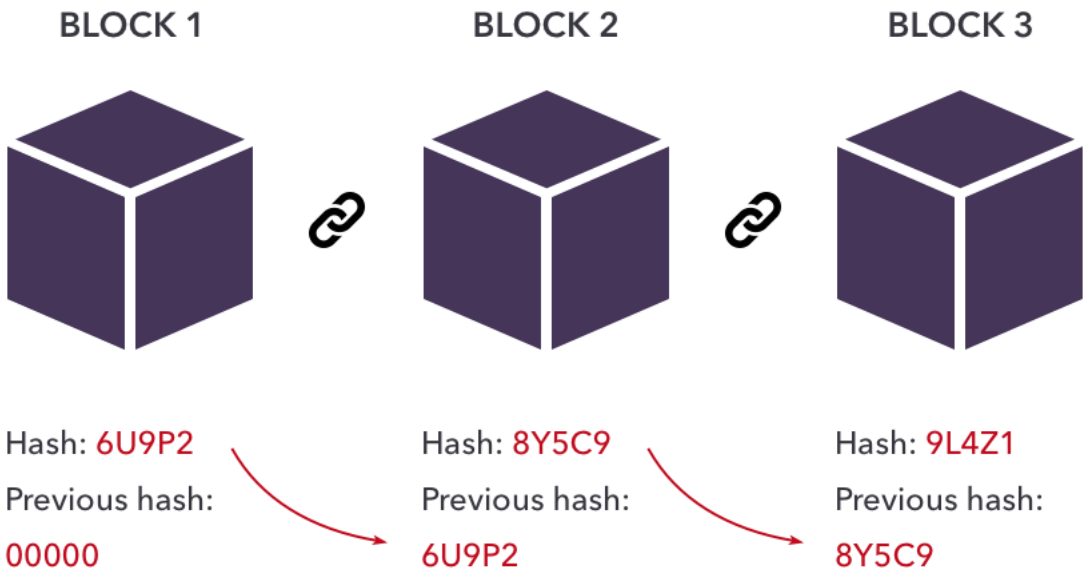
Structuri de date și Algoritmi – CD

PHANTOM GHOSTDAG

Tema 3

Deadline: 27.05.2022

Responsabil: Mihai Nan (*mihai.nan@upb.ro*)



Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2021 - 2022

Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să implementeze şi să utilizeze grafuri în rezolvarea unei probleme;
- să înţeleagă noțiunea de graf orientat aciclic;
- să implementeze un algoritm care permite obținerea informațiilor despre un registru de tranzacții;
- să transpună o problemă din viaţa reală într-o problemă care uzitează grafuri.

1 Descriere

În octombrie 2008, care era momentul de vârf al crizei financiare la nivel global, Satoshi Nakamoto (pseudonim folosit de o persoană sau de un grup de persoane) a publicat un document care definea cadrul unei monede digitale eliberată de orice autoritate legală: **Bitcoin** [1]. În cadrul acestei teme, veți înțelege în ce constă conceptul cheie, care a stat la baza tehnologiei folosită de această nouă monedă, denumit **Blockchain**. Conceptul de bază al acestui sistem este o rețea deschisă și anonimă de noduri, sau mineri, care împreună mențin un registru public al tranzacțiilor. Acest registru al tranzacțiilor este codificat sub forma unui lanț de blocuri (blockchain) în care fiecare bloc conține o serie de tranzacții noi ce au fost colectate de la utilizatori. În cadrul acestei teme, vom lucra cu conceptul BlockDAG care a fost introdus de Yonatan Sompolinsky, Shai Wyborski și Aviv Zohar în anul 2021 [2] și care reprezintă o generalizare a conceptului de blockchain propus de Satoshi Nakamoto. În cadrul acestui protocol, registrul de tranzacții este codificat sub forma unui Graf Orientat Aciclic.

1.1 Noțiuni generale

Pentru a prezenta noțiunile elementare de care avem nevoie pentru rezolvarea temei, vom porni de la un exemplu de graf orientat aciclic ce codifică o mulțime de blocuri. Acest exemplu este prezentat în Figura 1 și conține un bloc de geneză (blocul de pornire) ce este evidențiat cu albastru și alte blocuri normale ce conțin tranzacții ale utilizatorilor și sunt evidențiate cu galben.

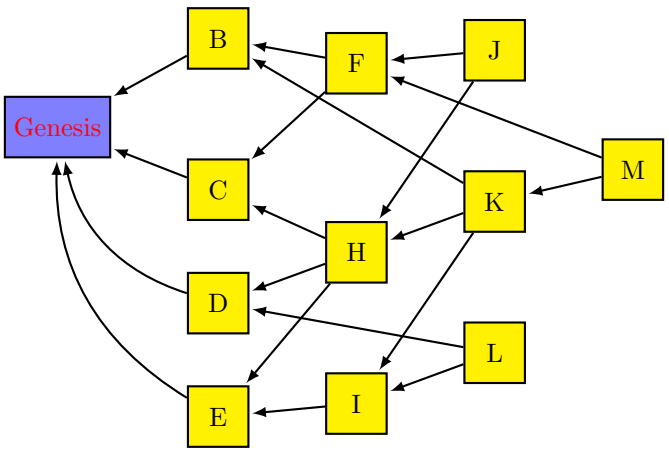


Figure 1: Exemplu de graf orientat aciclic ce descrie un registru de tranzacții

Fiecare bloc din graf poate avea unul sau mai multe blocuri precedente. Astfel, dacă blocul F le are ca blocuri precedente pe B și C, atunci o să apară în graful orientat arcele (F, B) și (F, C). Blocul de geneză este unul special și nu va avea niciun bloc precedent.

Vom mai defini următoarele concepte:

- `past(Node)` – reprezintă mulțimea cu blocurile referite direct sau indirect de către blocul `Node` și care trebuie să fie aplicate înainte de a fi aplicat blocul corespunzător lui `Node`.
- `future(Node)` – reprezintă mulțimea cu blocurile care au referință directă sau indirectă către blocul `Node` și care pot fi aplicate abia după ce blocul `Node` a fost aplicat.

- `anticone(Node)` – reprezintă mulțimea cu blocurile pentru care ordinea dintre ele și blocul `Node` este ambiguă. Găsirea unui algoritm de consensus care să decidă asupra ordinii dintre blocul `Node` și blocurile din mulțimea `anticone(Node)` reprezintă principala provocare.

- `tips(G)` – reprezintă mulțimea cu blocurile din graful G care au gradul intern egal cu 0 (nu există încă alte blocuri care să le aibă drept referință).

Pornind de la aceste definiții, pentru graful din Figura 1 și blocul `H` o să obținem următoarele mulțimi:

- `past(H) = {Genesis, C, D, E}`;
- `future(H) = {J, K, M }`;
- `anticone(H) = {B, F, I, L }`;
- `tips(G) = { J, L, M}` (unde `G` reprezintă întregul graf).

Definiție 1 Dându-se un graf orientat aciclic $G = (V, E)$, unde V reprezintă mulțimea nodurilor din graf și E reprezintă mulțimea arcelor din graf, o submulțime $S \subseteq V$ este numită **k-cluster** dacă pentru orice nod $B \in S$ avem îndeplinită relația $|anticone(B) \cap S| \leq k$.

Definiție 2 Numim **maximum k-cluster**, notăm cu MCS_k, pentru graful $G = (V, E)$ o submulțime $S^* \subset V$ de dimensiune maximă astfel încât $|anticone(B) \cap S^*| \leq k, \forall B \in S^*$.

Pentru a determina **maximum k-clusterul** pentru un graf $G = (V, E)$ Sompolinsky *et al.* [2] au propus următorul algoritm care determină acest cluster maximal (denumit $BlueSet$ și mulțimea $V \setminus BlueSet$).

Algorithm 1 Ordering the DAG

Require: G – a block DAG, k – the propagation parameter
Ensure: $BLUE_k(G)$ – the Blue set of G, ord – an ordered list containing all blocks in G

1: **procedure** ORDERDAG($G = (V, E), k$)
2: **if** $G == \{genesis\}$ **then**
3: **return** $\{genesis\}, \{genesis\}$
4: **end if**
5: **for** $B \in tips(G)$ **do**
6: $BlueSet_B, OrderedList_B \leftarrow$ **OrderDAG**$(past(B), k)$
7: **end for**
8: $B_{max} \leftarrow \arg\max \{|BlueSet_B| : B \in tips(G)\}$ ▷ break ties according to lowest name
9: $BlueSet_G \leftarrow BlueSet_{B_{max}}$
10: $OrderedList_G \leftarrow OrderedList_{B_{max}}$
11: add B_{max} to $BlueSet_G$
12: add B_{max} to the end of $OrderedList_G$
13: **for** $B \in anticome(B_{max}, G)$ **do** ▷ in some topological ordering
14: **if** $BlueSet_G \cup \{B\}$ is a k-cluster **then**
15: add B to $BlueSet_G$
16: **end if**
17: add B to the end of $OrderedList_G$
18: **end for**
19: **return** $BlueSet_G, OrderedList_G$
20: **end procedure**

Observații

Atunci când iterăm prin mulțimea `tips(G)` la linia 5 din algoritm, trebuie să ne asigurăm că mulțimea respectivă este sortată descrescător folosind următorul criteriu de comparare: vom sorta descrescător blocurile în funcție de cardinalul mulțimii `past`, iar în cazul blocurilor ce au număr egal de elemente în mulțimea `past`, vom realiza o sortare descrescătoare după nume.
Atunci când iterăm prin mulțimea `anticome(`B_{max}`, G)` la linia 13 din algoritm, trebuie să ne asigurăm că mulțimea respectivă este sortată crescător folosind următorul criteriu de comparare: vom sorta crescător blocurile în funcție de cardinalul mulțimii `past`, iar în cazul blocurilor ce au număr egal de elemente în mulțimea `past`, vom realiza o sortare crescătoare după nume.

Important

După cum putem observa, în algoritmul anterior se modifică graful furnizat ca parametru de la un apel recursiv la altul. Instrucțiunea `past(B)` va determina subgraful lui $G = (V, E)$ care conține doar nodurile care fac parte din mulțimea cu blocurile referite direct sau indirect de către blocul `B`.

$$G_{past}(B) = (past(B), \{(u, v) \in E | u \in past(B) \wedge v \in past(B)\})$$

Observații

În implementarea temei, mulțimea `OrderedList` folosită în algoritmul prezentat poate să fie ignorată. În cazul în care nu doriți să folosiți algoritmul propus pentru determinarea clusterului maximal, puteți porni de la propria propunere de algoritm.

2 Cerințe

2.1 Cerința 1

Pentru această cerință va trebui să se modeleze un registru de tranzacții sub forma unui graf reprezentat prin liste de adiacență și să se verifice dacă graful rezultat este unul valid.

În fișierul `blockdag.in` este oferită o reprezentare a unui registru de tranzacții, sub formă de text. Pornind de la această reprezentare, se va produce graful asociat, apoi se va verifica acest graf este un graf orientat aciclic. Dacă graful conține cel puțin un ciclu, atunci se va afișa mesajul `impossible`, iar dacă graful nu conține niciun ciclu, atunci se va afișă mesajul `correct` în fișierul `blockdag.out`.

Important

Este obligatoriu ca implementarea acestei cerințe să fie realizată utilizând o structură de date de tip graf orientat reprezentat prin liste de adiacență.
Orice alt tip de implementare **NU** va fi punctat.
Pentru a verifica dacă un graf orientat este sau nu aciclic, puteți folosi unul dintre algoritmii de parcurgere discutați în cadrul cursului.

2.2 Cerința 2

Pornind de la conceptele definite în Secțiunea 1.1, pentru această cerință va trebui să fie implementate funcții care să determine mulțimile `past(Node)`, `future(Node)`, `anticone(Node)` și `tips(G)`.

În fișierul `blockdag.in` este oferită o reprezentare a unui registru de tranzacții, sub formă de text. Pornind de la această reprezentare, se va produce graful asociat, apoi se vor determina blocurile din mulțimile cerute. Fiecare mulțime se va afișa pe un rând nou în fișierul `blockdag.out`. Pentru fiecare mulțime se vor afișă nodurile care fac parte din mulțime sortate în ordine crescătoare și cu un spațiu între ele. Pe prima linie se vor afișa blocurile care fac parte din mulțimea `past(Node)` (primul bloc afișat o să fie `Genesis` și apoi restul sortate crescător). Pe a doua linie se vor afișa blocurile care fac parte din mulțimea `future(Node)` sortate crescător și separate printr-un spațiu. Pe a treia linie se vor afișa blocurile care fac parte din mulțimea `anticone(Node)` sortate crescător și separate printr-un spațiu. Pe ultima linie se vor afișa blocurile care fac parte din mulțimea `tips(G)` sortate crescător și separate printr-un spațiu.

2.3 Cerință 3

Pentru această cerință va trebui să implementat și rulat algoritmul **OrderDAG** (0). Ne interesează doar să determinăm nodurile care fac parte din clusterul maximal.

În fișierul `blockdag.in` este oferită o reprezentare a unui registru de tranzacții, sub formă de text. Pornind de la această reprezentare, se va produce graful asociat, apoi se va determina **maximum k-clusterul** (în funcție de valoarea parametrului k ce o să fie furnizat sub formă de argument în linia de comandă). Rezultatul determinat se va afișa în fișierul `blockdag.out` în felul următor: numele nodurilor din clusterul maximal se

vor scrie pe aceeași linie din fișier separate printr-un unic spațiu. Primul nod o să fie cel corespunzător blocului de geneză, iar restul se vor afișa în ordine lexicografică.

Pentru o înțelegere mai bună a acestui algoritm, vom prezenta rezultatul rulării algoritmului pentru graful prezentat în Figura 1 și $k = 3$.

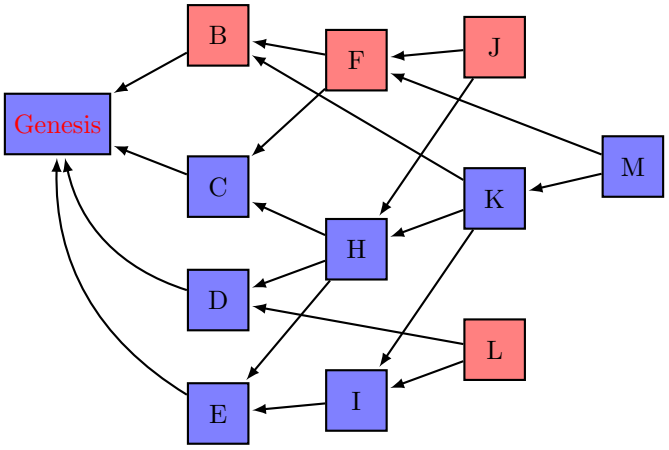


Figure 2: Rezultatul aplicării algoritmului OrderDAG pe graful oferit ca exemplu

Pornim de la acest rezultat și definițiile anterioare pentru a verifica dacă este corect, conform definițiilor oferite. În cazul acestui exemplu, $S = \{Genesis, C, D, E, H, I, K, M\}$.

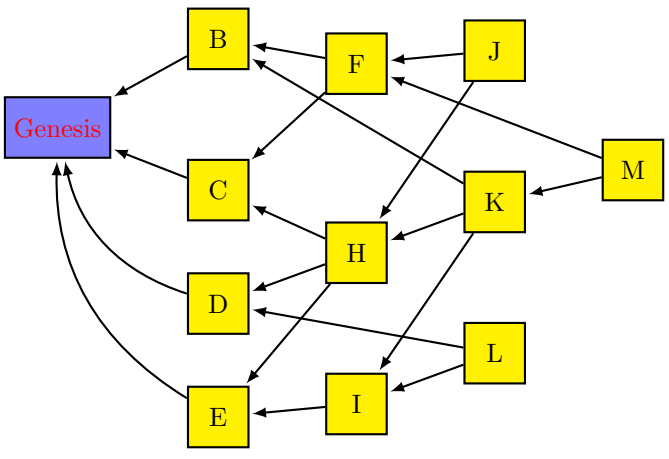
- $anticome(Genesis) \cap S = \emptyset$
- $anticome(C) \cap S = \{B, D, E, I, L\} \cap \{Genesis, C, D, E, H, I, K, M\} = \{D, E, I\}$
- $anticome(D) \cap S = \{B, C, E, F, I\} \cap \{Genesis, C, D, E, H, I, K, M\} = \{C, E, I\}$
- $anticome(E) \cap S = \{B, C, D, F\} \cap \{Genesis, C, D, E, H, I, K, M\} = \{C, D\}$
- $anticome(H) \cap S = \{B, F, I, L\} \cap \{Genesis, C, D, E, H, I, K, M\} = \{I\}$
- $anticome(I) \cap S = \{B, C, D, F, H, J\} \cap \{Genesis, C, D, E, H, I, K, M\} = \{C, H, D\}$
- $anticome(K) \cap S = \{F, J, L\} \cap \{Genesis, C, D, E, H, I, K, M\} = \emptyset$
- $anticome(M) \cap S = \{J, L\} \cap \{Genesis, C, D, E, H, I, K, M\} = \emptyset$

3 Formatul fișierelor

3.1 Fișier de intrare

Un exemplu de fișier de intrare este următorul:

```
12
Genesis B C D E F H I J K L M
Genesis :
B : Genesis
C : Genesis
D : Genesis
E : Genesis
F : B C
H : C D E
I : E
J : F H
K : B H I
L : D I
M : F K
```



Formatul acestui fișier este următorul:

- pe prima linie se găsește un număr natural n (n – numărul de noduri din graf);
- pe a doua linie se găsesc denumirile nodurilor din graf (fiecare nod va avea ca nume un șir de caractere de cel mult 10 caractere) despărțite prin câte un spațiu;
- pe următoarele n linii se vor găsi arcele grafului
  - fiecare linie este de forma `nume_nod : descendent_1 descendent_2 ... descendent_n`

Important

Numele nodurilor nu vor conține spații.
Grafurile din testele pentru cerințele 2 și 3 vor fi **grafuri orientate aciclice**.
Blocul de geneză va avea tot timpul numele `Genesis`.

3.2 Fișier de ieșire

Cerința 1

Fișierul de ieșire, corespunzător celui de intrare prezentat anterior ca exemplu, pentru prima cerință va avea conținutul:

```
correct
```

Cerința 2

Fișierul de ieșire, corespunzător celui de intrare prezentat anterior ca exemplu, pentru a doua cerință va avea următorul conținut (dacă se va apela pentru nodul `H`):

```
past(H) : Genesis C D E
future(H) : J K M
anticone(H) : B F I L
tips(G) : J L M
```

Cerința 3

Fișierul de ieșire, corespunzător celui de intrare prezentat anterior ca exemplu, pentru a treia cerință va avea următorul conținut (dacă se va apela cu valoarea $k = 3$):

```
Genesis C D E H I K M
```

4 Restricţii şi precizări

Temele trebuie să fie încărcate pe vmchecker. **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.

O rezolvare constă într-o arhivă de tip **zip** care conţine toate fişierele sursă alături de un **Makefile**, ce va fi folosit pentru compilare, şi un fişier **README**, în care se vor preciza detaliile implementării.

Makefile-ul trebuie să aibă obligatoriu regulile pentru **build** şi **clean**. Regula `build` trebuie să aibă ca efect compilarea surselor şi crearea binarului **blockdag**.

Programul vostru va primi, ca argumente în linia de comandă, o opţiune, pentru fiecare cerinţă în parte, în felul următor:

pentru cerința 1: `./blockdag -c1`

pentru cerința 2: `./blockdag -c2 Node`

pentru cerința 3: `./blockdag -c3 k`

Cerinta	Punctaj
Cerinţa 1	30 de puncte
Cerinţa 2	30 de puncte
Cerinţa 3	30 de puncte
Codying style, README, warning-uri	10 puncte

5 Punctaj

Atenţie!

Orice rezolvare care nu conţine structurile de date specificate nu este punctată.
Temele vor fi punctate doar pentru testele care sunt trecute pe vmchecker.
Nu lăsaţi warning-urile nerezolvate, deoarece veţi fi depunctaţi.
Dealocaţi toată memoria alocată pentru reţinerea informaţiilor, deoarece se vor depuncta pierderile de memorie.
Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.

Referințe

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review, page 21260, 2008.

[2] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom ghostdag: a scalable generalization of nakamoto consensus: September 2, 2021. In Proceedings of the 3rd ACM Conference on Advances in Financial Technologies, pages 57–70, 2021.