Tema 3 - Academia Network

Responsabili:

- Armand Nicolicioiu [mailto:armand.nicolicioiu@gmail.com]
- Teodor Duţu [mailto:teodor.dutu@gmail.com]

Data publicării : 24 aprilie, ora: 04:20

Deadline hard: 21 mai, ora 23:55

Deadline soft: 18 mai, ora 23:59. Se acordă un bonus de 10% din punctajul obținut pe temă echipelor care încarcă tema pe vmchecker până la aceasta dată.

Modificări şi actualizări

1 mai 2020 20:16: Adăugare schelet, teste şi checker

1 mai 2020 20:38: Prelungire deadline

1 mai 2020 23:37: Precizare `gcc-multilib`

2 mai 2020 00:00: Adăugare instanţă de upload pe vmchecker

3 mai 2020 15:09: Precizare despre modul în care trebuie încarcată arhiva

4 mai 2020 00:41: Completare condiție de egalitate get_oldest_influence

4 mai 2020 01:01: Precizare despre cross-citations în get_reading_order

4 mai 2020 01:08: Precizare limită distanță pentru find_best_coordinator

5 mai 2020 22:05: Plasare corectă a flagului `-lm` în `Makefile_tema3`

8 mai 2020 19:31: Checkerul parsează fisierele `.json` mai rapid.

15 mai 2020 05:41: Corectare teste şi încă o prelungire de deadline.

18 mai 2020 22:15: Corectare cerința 7. (cea discutată pe forum)

21 mai 2020 03:00: Completare condiție de egalitate find_best_coordinator

Obiective

- Exersarea și aprofundarea tuturor noțiunilor întâlnite la cursul de Structuri de Date.
- Accent pe întelegerea grafurilor și dezvoltarea capacității de a modela probleme ce au implicit o structură de graf.
- Reprezentarea și salvarea structurată a unor date reale pentru a permite extragerea eficientă a unor informații.
- Dezvoltarea abilității de a decide ce soluție răspunde cel mai bine cerințelor date într-o situație reală; tema nu impune folosirea unei structuri anume, ci oferă posibilitatea de explorare și analiză.
- Implementarea unui sistem care răspunde online la query-uri, datele pe care lucrează putând fi updatate între query-uri consecutive.
- Exersarea lucrului colaborativ prin rezolvarea temei în echipe de 2 studenți și folosirea tool-urilor de versionare.
- Trecerea de la soluții single file și implementări from stratch și obișnuirea cu lucrul pe un framework deja existent și întelegerea documentațiilor.

Introducere

Se va implementa un sistem ce agregă informații despre artcole științifice și răspunde la query-uri folosind cele mai recente date salvate. În timpul execuției programului se vor primi update-uri și query-uri intercalate.

Elementul central - articol științific

Tema va avea la bază articole științifice publicate la diferite conferințe. Un astfel de articol are urmatoarele proprietati:

- Titlu
- ID unic
- Anul publicării
- Conferința la care a fost publicat (string unic)
- Domenii de studiu
  - o listă de string-uri. ex: {“Computer Vision”, “Machine Learning”}
  - reprezintă domeniile pe care le studiază articolul curent
- Lista autorilor
  - Pentru fiecare autor din listă se primesc următoarele informații
    - Nume autor
    - ID autor
    - Instituția asociată
      - Exemple: “Pantelimon George Popescu, 2805920242, Politehnica University of Bucharest” și “Robert Tarjan, 2089263920, Princeton University”
- Lista articolelor citate (references)
  - La finalul unui articol științific se trec sursele pe care s-a bazat pentru a susține ipoteze și din care au fost folosite idei. Putem spune că articolul curent “este influențat” sau “depinde de” toate articolele din lista de references.
  - Această listă de articole citate conține id-urile altor articole științifice care la rândul lor sunt entități de tipul descris în acest paragraf și la rândul lor vor cita alte articole.

Descriere și mod de lucru

În arhiva temei este prezentat un API (descris de fișierul header publications.h) ce va trebui implementat de voi și va fi apelat de către checker pentru a verifica corectitudinea. Acesta conține:

- O singură funcție de tip “input” ce primește ca parametri toate informațiile ce descriu un singur articol.
  - Sunteți responsabili de definirea și implementarea propriilor structuri în care salvați acest input și de a gândi arhitectura soluției astfel încât să rezolvați cerințele cât mai eficient.
  - Această funcție doar pasează programului vostru input-ul, iar voi decideți cum și unde veți salva aceste informații. Pentru a nu fi nevoiți să apelați la variabile globale, fiecare apel din API va primi și pointer către o structură auxiliară.
  - Observații
    - Toate datele necesare implementării temei vor fi salvate în structura `PublData`, declarată în `publications.h` și pe care o veți implementa (îi veți declara membrii) în alt fișier (de exemplu `publications.c`) astfel încât un cod care include `publications.h` să nu aibă acces la membrii acestei structuri în mod direct. Vedeți exemplul de mai jos pentru detalii.
    - Un articol poate avea printre referințe articole care nu s-au citit încă. Acest lucru este cauzat de faptul că în general, sistemele care înregistrează articole nu garantează că le vor înregistra în ordine cronologică.

        - Este posibil ca doua articole să se citeze reciproc. Acest lucru se poate întâmpla când două articole sunt scrise in paralel, iar versiunile lor intermediare sunt disponibile public, putând fi citate deja de alte surse. La final, două articole ajung să se citeze reciproc. Altă situație poate fi cand articolele sunt scrise de aceiași autori și trimise la aceeași conferință.

- Mai multe funcții de tip “query”.
    - Acestea primesc anumiți parametri ce descriu cerința și au ca valoare de return tipul de date așteptat de checker.
    - Urmează să fie detaliată fiecare în parte la paragraful “Cerințe”.
    - Programul vostru se va baza pe informaţia salvată în structura `PublData` în care aveți structurată informația pasată cu ajutorul comenzii `add_paper` și trebuie doar să întoarcă outputul cerut. Nu va fi nevoie să faceți niciun fel de printuri. De altfel, se recomandă să nu le faceți decât pentru debug, după care să le eliminați, deoarece orice formă de `I/O` încetineşte procesul.
    - Se pot rezolva în orice ordine, cerințele fiind independente și existând teste individuale.

Structura PublData

În această structură va trebui să stocați toate datele pe care le veți folosi pentru rezolvarea cerințelor de mai jos. Structura este declarată în `publications.h`. Voi va trebui să îi definiți membrii în `publications.c` sau în orice alt fişier, cu condiția ca acești membri să nu fie vizibili din afara API-ului vostru (structura să fie “privată”).

Mai precis, trebuie să nu puteți accesa direct membrii structurii din afara API-ului. Presupunând că există în cadrul structurii voastre membrul int ceva, nu ar trebui să puteți compila codul următor, dacă acesta este scris în afara API-ului:

Aşa NU

```
PublData* data = init_publ_data();  // functia este descrisa mai jos
data->ceva = 1;  // aici ar trebui sa apara eroarea de compilare
```

Pentru structura `PublData`, va trebui să implementați două funcții care vor aloca, respectiv dealoca memoria utilizată pentru aceasta.

Alocarea structurii - init_publ_data

Semnătură:

```
PublData* init_publ_data(void)
```

Funcția va aloca o structura de tip `PublData` și va întoarce un pointer la aceasta. Este la latitudinea voastră dacă faceți inițializările aici sau pe măsură ce se execută funcțiile din API. Această funcție este apelată `runner.c` înainte să se înceapă rularea funcțiilor propriu-zise ale API-ului.

Dealocarea structurii - destroy_publ_data

Semnătură:

```
void destroy_publ_data(PublData* data)
```

Funcția dealoca toata memoria folosită de variabila data. În runner.c, aceasta functie se apelează după ce se apelează toate funcțiile din API.

Cerințe

Toate functiile care returneaza vectori vor trebui sa aloce memorie separta pentru acestia, intrucat vectorii vor fi

dealocati in `runner.c`.

Cerința 0 - add_paper

Semnătură:

```
void add_paper(PublData* data, const char* title,
    const char* venue, const int year, const char** author_names,
    const int64_t* author_ids, const char** institutions,
    const int num_authors, const char** fields, const int num_fields,
    const int64_t id, const int64_t* references, const int num_refs);
```

Funcția primește ca parametri toate datele referitoare la un articol (titlu, jurnalul la care a fost publicat, anul apariției, numele autorilor, instituțiile din care aceştia fac parte, domeniile de studiu în care se încadrează articolul, id-ul acestuia, precum și lista articolelor pe care acesta le citează) şi adaugă acest articol în logica de articole reținută în structura `PublData`.

Cerința 1 - get_oldest_influence

Semnătură:

```
char* get_oldest_influence(PublData* data, const int64_t paper);
```

Funcția primește ca parametru id-ul unui articol și se dorește găsirea celui mai vechi (cu cel mai mic an al aparitiei) articol de care acesta “depinde”. Spunem că articolul X depinde de articolul Y dacă **una** din condițiile de mai jos este îndeplinită:

- Y se află în lista de citări (references) a lui X.
- Cel puțin un articol citat de X depinde de Y.

Observăm că avem o **definiție recursivă**.

Dacă există mai multe articole cu același an al apariției şi satisfac cerințele anterioare, se va întoarce cel cu numărul maxim de citări. Dacă în continuare există egalitate, se va întoarce cel cu id-ul minim.

în timpul căutării se vor ignora articolele citate dar pe care sistemul nostru nu le-a primit incă printr-un apel de `add_paper`. Dacă pentru un articol dat nu se găsește niciun alt articol care să indeplinească cerințele, funcția va întoarce string-ul `“None”`.

Cerința 2 - get_venue_impact_factor

Semnătură:

```
float get_venue_impact_factor(PublData* data, const char* venue);
```

Funcția întoarce factorul de impact al jurnalului primit ca parametru. Factorul de impact se defineşte ca fiind numărul mediu de citări ale tuturor articolelor publicate în jurnalul dat până în momentul în care se primește query-ul. Rezultatul poate să difere de cel din fișierele de referință cu maximum `0.001`.

La fel ca la cerința precedentă, implementarea nu va ține cont de articole citate dar care încă nu au fost adăugate în sistem. De exemplu, articolul X poate aparea pe lista de references a unui articol Y deja adăugat, insă știm doar id-ul lui X, încă nu știm restul informațiilor despre el, printre care și jurnalul la care a fost publicat.

Cerința 3 - get_number_of_influenced_papers

Semnătură:

```
int get_number_of_influenced_papers(PublData* data, const int64_t id_paper, const int distance);
```

Funcția primește id-ul unui articol și o anumită distanță și folosește aceeași definiție pentru X este influențat de Y ca la Cerința 1. Se cere numărul total de articole care au fost influențate de articolul dat. Practic, Y a influențat X și Z. X și Z au influențat la rândul lor alte articole, etc. Toate acestea formează mulțimea articolele influențate de Y și trebuie să fie numărate.

Parametrul “distance” are rolul de a limita distanța de la Y la un articol care depinde de el. Mai exact, numărăm doar articolele care depind de Y și se află la o distanță cel mult “distance” în graful de citări.

Cerința 4 - get_erdos_distance

Semnătură:

```
int get_erdos_distance(PublData* data, const int64_t id1, const int64_t id2);
```

Se primesc id-urile a doi autori și se dorește calcularea distanței Erdos dintre cei doi. Definim distanța Erdos în felul următor:

Dacă autorul A a publicat un articol împreună cu autorul B, înseamnă că distanța dintre ei este minimă, adică 1. Dacă la rândul lui B a publicat un articol cu C (iar A nu a publicat cu C) distanța Erdos dintre A și C este 2, pentru că se află la 2 muchii distanță în graful de autori, unde o muchie între doi autori înseamnă că au publicat un articol împreună. Analog pentru distanțe mai mari.

Intuiție: “cunosc pe cineva, care cunoaște pe cineva, care cunoaște … etc” sau “vărul unchiului mătușii mamei lui” sau, mai simplu, gradul la care ești conectat cu cineva pe linkedin.

Disclaimer: e adevărat că distanța Erdos nu se defineşte în realitate ca mai sus, ci ca fiind față de un singur autor, şi anume Paul Erdos [https://en.wikipedia.org/wiki/Erd%C5%91s_number], de la care îi provine și numele. Cerința dată este o generalizare.

Dacă nu se găsește niciun drum între cei doi autori dați, funcția va retuna `-1`

Cerința 5 - get_most_cited_papers_by_field

Semnătură:

```
char** get_most_cited_papers_by_field(PublData* data, const char* field, const int* num_papers);
```

Se primesc numele unui domeniu și numărul de articole dorit. Se vor întoarce **titlurile** celor mai citate `num_papers` care studiază acel domeniu, adică `field` se află în lista lor de domenii studiate. **Un articol X numără câte o citare pentru fiecare articol Y care în lista lui de references conține articolul X.**

În cazul în care exista exista mai multe articole cu același număr de citări, se vor returna titlurile celor mai recente (cu anul de apariție maxim). Dacă articolele în cauză au apărut în același an (sunt la fel de recente), criteriul de departajare va fi id-ul (id cât mai mic).

Output-ul acestei funcții este de tipul `char**`. Reprezintă un array de string-uri ce va fi alocat dinamic în funcție și în care se vor copia titlurile așteptate. Runnerul va citi acest răspuns, apoi se va ocupa de eliberarea memoriei.

Parametrul `*num_papers` are un rol dublu: inițial indică numarul de artcole ce se doreste intors. In cazul in care lista intoarsa contine mai putine elemente, acest numar se va suprascrie cu dimensiunea listei intoarse.

Cerința 6 - get_number_of_papers_between_dates

Semnătură:

```
int get_number_of_papers_between_dates(PublData* data, const int early_date, const int late_date);
```

Funcția va calcula si va returna câte articole au fost publicate în total între cei doi ani primiți ca parametru.

Cerința 7 - get_number_of_authors_with_field

Semnătură:

```
int get_number_of_authors_with_field(PublData* data, const char* institution, const char* field);
```

Se primesc numele unei instituții și un domeniu de studiu. Să se determine numărul total de autori care au publicat articole despre acel domeniu în timp ce erau asociați cu instituția dată.

Cerința 8 - get_histogram_of_citations

Semnătură:

```
int* get_histogram_of_citations(PublData* data const int64_t id_author, int* num_years);
```

Se dorește calcularea numărului de citări ale tuturor articolelor publicate de autorul primit ca parametru. Funcția returnează aceasta histograma sub forma unui vector de lungime `num_years` (valoare pe care tot funcția o va seta). Acest vector contine pe poziția `i`, numărul de noi citări ale articolelor autorului cu `i` ani în urma fata de anul curent (2020). Astfel, pe poziția 0 va fi numărul de citări din 2020, pe poziția 1, cel din 2019 s.a.m.d.

Output-ul acestei funcții este de tipul `int**`. Reprezintă un array de intregi ce va fi alocat dinamic în funcție în care se vor scrie valorile așteptate. Runnerul va citi acest răspuns, apoi se va ocupa de eliberarea memoriei. Parametrul `int* num_years` este de tip “output” - la adresa indicată se va scrie dimensiunea histogramei întoarse

Cerința 9 - get_reading_order

Semnătură:

```
char** get_reading_order(PublData* data, const int64_t id_paper, const int distance, int* num_papers);
```

Se primește un articol care se dorește a fi citit și o anumită distantă. Considerăm că pentru a înțelege mai bine un articol, trebuie să citim întâi articolele pe care le-a citat, analog pentru acestea. Practic, înainte de a citi un articol, dorim să citim toate articolele de care acesta depinde / de care este influențat. Să se găsească o ordine în care aceste articole să fie citite astfel încât să se respecte criteriile de mai sus. Pentru o distanță finită specificată, se consideră doar articolele de care depinde articolul curent și se află la o distanță mai mică sau egală cu `distance` față de acesta.

În cazul în care există mai multe soluții posibile vom aplica și următoarele criterii, în ordine:

1. Dacă anumite articole pot fi citite în orice ordine fără a contrazice criteriile enunțate anterior, acestea se vor citi în ordinea dată de anul apariției.
2. Dacă în continuare există anumite articole care pot fi citite în orice ordine, se vor citi în ordine crescătoare a id-urilor.

Respectând aceste două criterii adiționale, output-ul va fi unic determinat.

Output-ul acestei funcții este de tipul `char**`. Reprezintă un array de string-uri ce va fi alocat dinamic în funcție și în care se vor copia titlurile așteptate. Runnerul va citi acest răspuns, apoi se va ocupa de eliberarea memoriei. Parametrul `int* num_papers` este de tip “output” - la adresa indicată se va scrie dimensiunea listei întoarse.

În cazul acestei cerințe, testele au fost construite încât să nu existe situația *X depinde de Y și Y depinde de X*. Astfel, ordinea dorită se poate defini.

Cerința 10 - find_best_coordinator

Semnătură:

```
char* find_best_coordinator(PublData* data, int64_t id_author);
```

Un student a publicat câteva articole și vrea să aplice la doctorat. Pentru a face acest lucru, trebuie să găsească un

profesor coordonator de doctorat cât mai potrivit. Pentru fiecare autor din sistem (altul decât studentul dat), vom defini scorul următor:

$$score_i = e^{-erdos(a,a_i)} \sum_{j=1}^{n_i} num\_citations_j * impact\_factor_j$$

Unde:

- $score_i$ este scorul calculat pentru un anumit autor a_i.
- a este autorul dat ca parametru funcției.
- n_i este numărul de articole scrise a_i.
- $num\_citations_j$ este numărul de citări ale articolului j al autorului i.
- $impact\_factor_j$ este factorul de impact al jurnalului la care a fost publicat articolul j al autorului i.

Să se întoarcă numele autorului care maximizeaza acest scor.

Intuiție: A fost aleasă o euristică [https://en.wikipedia.org/wiki/Heuristic_(computer_science)] ce mărește scorul dacă autorul găsit are publicații cu multe citări la conferințe prestigioase, dar scade scorul pe măsură ce distanța erdos dintre cei doi crește. Dacă sunt mai mulți autori cu același scor maxim, se va alege cel cu distanta erdos minimă. În caz că încă există egalitate, se alege cel cu id-ul minim."

Se vor considera doar autorii care se află la o distanță *erdos* în intervalul [1, 5] față de autorul dat.

Compilare si rulare

API - publications.o

Sarcina voastră este să implementați API-ul descris astfel încât la rularea comenzii `make build` să fie creat un fişier obiect, numit `publications.o`. Pentru aceasta, aveți pus la dispoziție makefile-ul `Makefile`. Aveți libertatea să-l modificați cum doriți, cu mențiunea că acesta trebuie să creeze fișierul `publications.o` la rularea comenzii make build și să aibă și o regulă clean care va șterge `publications.o` și orice alte fișiere binare care rezultă în urma compilării obiectului `publications.o`. Regula `clean` nu trebuie să șteargă și fișierele binare create de restul scheletului, care vor fi detailate mai jos.

Schelet

Infrastructura de testare este destul de amplă, pe de o parte pentru ca modulul `Parson` ce se ocupă cu parsarea fișierelor `.json` este destul de stufos, dar și pentru a vă obișnui cu ideea de a lucra într-un proiect mai mare, la care voi trebuie sa adăugați o funcționalitate anume (în acest caz, API-ul descris în enunț).

Scheletul si checkerul temei sunt disponibile aici: tema3_2020_skel_checker.zip

Rularea funcțiilor din API - runner

În `runner.c` se parsează câte un fişier de test și se apelează funcțiile din acesta, împreuna cu parametrii fiecăreia. Observați folosirea keywordului `static` pentru acele funcții care nu trebuie să fie vizibile din afara `runner.c`. Se recomandă folosirea acestui tip de funcții și în implementarea API-ului, acolo unde acest lucru este posibil (oriunde este vorba de o funcție care nu se apelează decât din interiorul API-ului, adică, în general orice funcție auxiliară vă definiții). Fiecare funcție statică din `runner` își va apela corespondenta din API și va compara outputul acesteia cu cel citit din fişierul de referință (în afară de `run_add_paper`, care apelează `add_paper`, care nu intoarce nimic).

Makefile_tema3

`Makefile_tema3` conține regulile `build` şi `clean`. Prima dintre acestea va crea executabilul `tema3`, iar

clean va șterge acest executabil, precum și toate fișierele binare rezultate în urma compilării.

Puteți folosi regulile din `Makefile_tema3` astfel:

```
make -f Makefile_tema3 <regula>
```

Executabilul `tema3` va linka `publications.o`, pe care `Makefile_tema3` îl creează folosind `make build`.

Dat fiind că, pentru a diminua din problemele care pot apărea ca urmare a faptului că mașinile voastre sunt pe 64 de biți, iar vmcheckerul este pe 32, ambele Makefile-uri folosesc flagul `-m32`. Pentru a putea să-l folosiți și voi, va trebui să vă instalați biblioteca `gcc-multilib` astfel:

```
sudo apt-get install gcc-multilib
```

Parserul pentru fișiere .json - Parson

Pentru a putea parsa un fișier de tip `.json`, așa cum sunt fișierele ce conțin articolele cu care se va opera, se folosește parserul `Parson`. În general, orice parser va citi tot fișierul și va crea o structură de date ce va conține toate datele din acel fișier. Această structura de date poate fi asemănată cu un hashtable, mapând numele câmpurilor din fișierul `.json`, ca stringuri, la valorile acestora. Acesta este și cazul parserului **Parson**:

- Structura principalală este `JSON_Object`. Acesta poate conține câmpuri (stringuri, numere etc.), array-uri (`JSON_Array`), precum și alte `JSON_Object`-uri.
- Parserul pune la dispoziție funcții care extrag date, fie din array-uri

```
json_array_get_<tip_de_date>(JSON_Array* array, const char* key)
```

fie din obiecte

```
json_object_get_<tip_de_date>(JSON_Object* object, const char* key)
```

- Funcțiile care întorc stringuri nu alocă memorie din nou pentru acele stringuri, ci întorc pointeri la datele stocate intern în `JSON_Array` sau `JSON_Object`. Din acest motiv, în funcția `run_add_paper` nu se dealocă şi parametrii pasați funcției `add_paper`, ci doar parserul.
- Probabil cea mai bună documentație se află chiar pe repository-ul parserului [https://github.com/kgabis/parson].

Arhiva

Trebuie sa încărcați pe vmchecker [https://elf.cs.pub.ro/vmchecker/ui/#SD] o arhivă `.zip` care să conțină fișierul `Makefile` care creează `publications.o`, sursele din care acest fişier obiect este creat, precum și `README`-ul care prezintă implementarea API-ului.

Un singur coechipier va încărca arhiva pe numele său pentru amandoi. In cazul in ambii coechipieri încărca tema, se va corecta cea mai recentă submisie.

Numele arhivei nu e important; oricum este suprascris cu id-ul vostru de pe Moodle [https://acs.curs.pub.ro], în momentul în care incărcați tema.

Puteți să modificați orice în cele doua fișiere `publications.c` și `publications.h` date, cu excepția semnăturilor funcțiilor din `publications.h` și a macro-ului `DIE` (pe care chiar vă recomandăm sa îl folosiți în implementarea temei).

Pentru mai multe detalii despre acest macro, puteți accesa acest link [https://ocw.cs.pub.ro/courses/so/laboratoare/resurse/die].

Nu includeți alte fișiere din schelet (directorul `checker` sau `Makefile_tema3`) în arhivă, pentru că vor fi suprascrise!

Dați `make -f Makefile_tema3 clean` înainte să creați arhiva! O arhivă care conține fișiere binare va primi o depunctare de 10p.

Punctaj

Atenție! O temă care nu compilează sau care nu trece niciun test va primi 0 puncte.

1. 90p teste: **fiecare** test este verificat cu valgrind. Dacă un test are memory leaks, nu va fi punctat.
2. 10p README: trebuie făcut explicit, cât să se înțeleagă ce ați făcut în sursă, dar fără comentarii inutile și detalii inutile. Aceste puncte vor fi acordate automat de către checker dacă există un fişier `README`/`README.txt`/`README.md` (formatul pe care vi-l recomandăm). Se pot aplica depunctari pentru `README`-uri care nu respectă cerințele.
3. 10p coding style: acesta este verificat automat folosind `cpplint.py` şi este **binar**: ori e bine şi primiți punctele, ori nu e bine şi nu primiți nimic
4. Temele au deadline hard. Prin urmare, o temă trimisă după deadline este punctată cu 0.

Nu copiați! Toate soluțiile vor fi verificate folosind o unealtă de detectare a plagiatului. În cazul detectării unui astfel de caz, atât plagiatorul cât și autorul original (nu contează cine e) vor primi punctaj 0 pe **toate temele**!

De aceea, vă sfătuim să nu vă lăsați rezolvări ale temelor pe calculatoare partajate pe mail/liste de discuții/grupuri etc.

FAQ

Q: Putem implementa tema în C++?
A: Din păcate, nu :(

Q: Putem include în publications.h biblioteca `<blablabla>.h`?
A: `<blablabla>.h` e un header, **NU** o bibliotecă. Biblioteca se cheamă libc [https://en.wikipedia.org/wiki/C_standard_library] şi este declarată în mai multe headere, printre care şi `<blablabla>.h`, pe care, da, aveți voie să-l includeți.

Q: Putem să “spargem” implementarea API-ului în mai multe fişiere sursă şi, eventual, mai multe headere?
A: Da.

Q: Putem folosi flaguri de optimizare în `Makefile`?
A: Nu.

Q: Putem folosi variabile globale?
A: Nu. Folosiți structura `PublData` sau, dacă din orice motiv nu vi se pare potrivit, folosiţi variabile **statice**.

Q: Am trimis tema cu *x* secunde după deadline. Mi se poate lua în considerare?
A: Nu.

Link-uri utile

https://www.aminer.org/citation [https://www.aminer.org/citation] - dataset-ul din care au fost extrase articolele din teste.