UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Implementarea unui algoritm de tip programare dinamică în Dafny ce rezolvă problema de selecție a activităților cu profit maxim

propusă de

Roxana Mihaela Timon

Sesiunea: februarie, 2024

Coordonator științific

Conf. Dr. Ciobâcă Ștefan

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Implementarea unui algoritm de tip programare dinamică în Dafny ce rezolvă problema de selecție a activităților cu profit maxim

Roxana Mihaela Timon

Sesiunea: februarie, 2024

Coordonator științific

Conf. Dr. Ciobâcă Ștefan

Avizat,

Îndrumător lucrare de licență,

Conf. Dr. Ciobâcă Ștefan.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Timon Roxana Mihaela** domiciliat în **România, jud. Vaslui, sat. Valea-Grecului, str. Bisericii, nr. 24**, născut la data de **09 iulie 2000**, identificat prin CNP **6000709375208**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2022, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Implementarea unui algoritm de tip programare dinamică în Dafny ce rezolvă problema de selecție a activităților cu profit maxim** elaborată sub îndrumarea domnului **Conf. Dr. Ciobâcă Ștefan**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Implementarea unui algoritm de tip programare dinamică în Dafny ce rezolvă problema de selecție a activităților cu profit maxim**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Roxana Mihaela Timon**

Data: ..... Semnătura: .....

Cuprins

Motivație

Am ales sa fac această temă deoarece Dafny era un limbaj de programare nou pentru mine si am considerat a fi o provocare. Știam doar că acesta este folosit pentru a asigura o mai mare siguranță si corectitudine, putând fi aplicat in industria aerospațiala, în industria medicala, la dezvoltarea sistemelor financiare, în securitate și criptografie. Totodată, prin demonstrarea de corectitudine a unui algoritm in Dafny puteam să-mi folosesc pe langă cunostințele informatice si pe cele de matematică, de care am fost mereu atrasă. Pentru a crește gradul de complexitate al lucrării am decis sa folosesc ca și tehnică de proiectare programarea dinamică. Astfel, având posibilitatea sa înțeleg mai bine cum funcționează programarea dinamică și să demonstrez că, cu ajutorul ei se obține o soluție optimă.

Intenție

În cadrul lucrării voi discuta despre limbajul de programare Dafny, surpinzând particularitățile sale, despre problema de selecție a activităților cu profit maxim, despre programarea dinamică și avantajele sale în comparație cu alte tehnici de proiectare a algoritmilor. Voi prezenta, de asemenea, demonstrația de corectitudine a problemei de selecție a activităților cu profit maxim folosind programarea dinamică în Dafny.

Introducere

Lucrarea este structură în 3 capitole:

- **Dafny**. În acest capitol voi prezenta particularitățile limbajului de programare Dafny și ariile de aplicabilitate.
- **Programarea dinamică**. În acest capitol voi reaminti despre programarea dinamică ca tehnică de proiectare a algoritmilor, despre avantajele sale în comparație cu alte tehnici de proiectare, precum Greedy.
- **Problema de selecție a activităților cu profit maxim**. Acest capitol conține prezentarea algoritmului de selecție a activităților cu profit maxim, demonstrația de corectitudine cu ajutorul limbajului de programare Dafny și tehnica de lucru abordată.

Capitolul 1

Dafny

1.1 Prezentare generală

Dafny este un limbaj imperativ de nivel înalt cu suport pentru programarea orientată pe obiecte. Metodele realizate în Dafny au precondiții, postcondiții și invarianți care sunt verificate la compilare, bazându-se pe soluționatorul SMT Z3. În cazul în care o postcondiție nu poate fi stabilită (fie din cauza unui timeout, fie din cauza faptului că aceasta nu este valabilă), compilarea eșuează. Prin urmare, putem avea un grad ridicat de încredere într-un program verificat cu ajutorul sistemului Dafny. Acesta a fost conceput pentru a facilita scrierea unui cod corect, în sensul de a nu avea erori de execuție, dar și corect în sensul de a face ceea ce programatorul a intenționat să facă.

1.2 Tipuri de date

În Dafny există mai multe tipuri de date care pot fi utilizate pentru a defini variabile și structuri de date:

- **int**: folosit pentru a declara numere intregi
  ex. var x: int := 10;
- **bool**: folosit pentru a declara valori de adevar: true si false
  ex. var esteAdevarat: bool := true;
- **char**: folosit pentru a declara caractere
  ex. var litera: char := 'A';

- **string** : folosit pentru a declara siruri de caractere
  ex. var mesaj: string := "Am terminat codul!";
- **seq**<**T**>: folosit pentru a declara secvente de elemente ordonate de tip T, precum liste, cozi si stive, fiind imutabile odata ce au fost create
  ex. var oSeq: seq<int> := {1, 2, 3};
- **set**<**T**>: folosit pentru a declara seturi de elemente unice de tip T
  ex. var unSet: set<int> := {1, 2, 3};

Acestea sunt doar cateva exemple, limbajul oferă suport si pentru structuri de date mai complexe, inclusiv tipuri de date definite de utilizator.

1.3 Metode , funcții si predicate

1.3.1 Metode

Metodele în Dafny reprezintă blocuri de intrucțiuni ce pot avea un comportament stabilit, apelate sau pentru a calcula valori, sau pentru a realiza anumite instrucțiuni sau ambele. Fiecare metoda poate fi caracterizată de precondiții si postcondiții care trebuiesc îndeplinite. Precondițiile trebuie să fie îndeplinite la apelarea metodei pentru ca aceasta să se execute, iar postcondițiile trebuie sa fie îndeplinite după ce se termină de executat corpul metodei. O metoda a caror postcondiții nu sunt îndeplinite nu poate fi demonstrată ca fiind corectă. Postcondițiile reprezinta ceea ce Dafny știe în urma apelului unei metode, el uitând corpul acesteia.
O metodă în Dafny este definită folosind cuvântul cheie ***'method'***.

```
method SumaCuVerificareM(a: int, b: int) returns (rezultat: int)
    ensures rezultat == a + b
{
    return a + b;
}
```

Cu ajutorul postconditiei *ensures rezultat == a + b* assert-ul urmator va putea fi evaluat cu true. Altfel, dacă ensures-ul ar fi lipsit, rezultatul funcției, respectiv *AdunaCuVerificareM(3,2)* nu poate fi știut.

```
method Main()
{
  var suma := AdunaCuVerificareM(3,2);
  assert suma == 5;
}
```

1.3.2 Funcții

Funcțiile, în schimb contin o singura instrucțiune și au ca scop calcularea unor funcții pur matematice. Spre deosebire de metode, Dafny nu uită corpul unei funcții atunci când acestea sunt apelate din exterior: alte metode, lemme. Funcțiile nu fac niciodată parte din programul final compilat, ele sunt doar instrumente care ne ajută să ne verificăm codul. O funcție în Dafny este definită folosind cuvântul cheie **'function'**.

```
function AdunaCuVerificareF(a: int, b: int): int
{
    a + b;
}
```

În acest caz, daca vom apela în *main()* funcția *AdunaCuVerificareF(3,2)* assert-ul va avea loc cu succes.

```
method Main()
{
  var suma := AdunaCuVerificareF(3,2);
  assert suma == 5;
}
```

1.3.3 Predicate

În Dafny, predicatele sunt utilizate pentru a specifica condiții sau proprietăți care trebuie să fie adevărate în anumite contexte, cum ar fi în invarianții buclelor sau postcondițiile metodelor. Predicatele sunt adesea exprimate ca expresii booleene și joacă un rol crucial în specificarea proprietăților de corectitudine. Predicatul de mai jos asigura că variabila n este număr par. Un predicat în Dafny este definit folosind cuvântul cheie **'predicate'**.

```
predicate EstePar(n: int)
{
    n % 2 == 0
}
```

1.4 Precondiții, poscondiții si invarianți de buclă

1.4.1 Precondiții, poscondiții

Precondițiile și postcondițiile reprezintă proprietăți care trebuie sa fie îndeplinite la intrarea (precondiții) și respectiv, la ieșirea (postcondiții) dintr-o metodă, lemă, funcție. Astfel, adevărata putere a Dafny-ului vine din capacitatea de a adnota metodele pentru a le specifica comportamentul. O precondiție în Dafny este definită folosind cuvântul cheie **'requires'**. O postcondiție în Dafny este definită folosind cuvântul cheie **'ensures'**.

```
method ElementNeutru(x: int, y: int) returns (suma: int)
  requires x == 0
  ensures suma == y
{
  returns x + y;
}
```

În metoda de mai sus, pentru ca suma sa fie egala cu valoarea lui y trebuie ca x sa respecte precondiția *requires x == 0*.

1.4.2 Invarianti de buclă

Un invariant de buclă este o expresie care se menține chiar înainte de testul buclei, adică la intrarea într-o buclă și după fiecare execuție a corpului buclei. Aceasta surprinde ceva care este invariabil, adică nu se schimbă, la fiecare pas al buclei.

```
var i := 0;
while i < n
  invariant 0 <= i
{
```

```
    i := i + 1;
}
```

Trebuie să demonstrăm că executarea corpului buclei înca o data face ca invariantul să fie valabil din nou. La fel cum Dafny nu va descoperi singur proprietățile unei metode, nu va ști că se păstrează alte proprietăți decât cele elementare ale unei bucle, decât dacă i se spune prin intermediul unui invariant. Cu alte cuvinte, dupa executarea corpului buclei, Dafny va sti doar proprietațile declarate cu ajutorul invarianților.

1.4.3 Terminarea buclei

Dafny trebuie sa demonstreze că o buclă while nu se execută la nesfârșit, prin utilizarea adnotărilor *decreases*. Există două locuri în care Dafny trebuie sa demonstreze terminarea: buclele și recursivitatea. Ambele situații necesită fie o adnotare explicită, fie o presupunere corectă din partea lui Dafny. Ca terminarea să poate fi demonstrată trebuie ca Dafny să verifice că expresia devine din ce în ce mai mică, și că aceasta are o limită inferioară.

1.5 Aplicabilitate

Asigurând încredere și corectitudine, Dafny poate fi folosit în numeroase industrii, precum:

- **industria aerospațială** (la navigație, comunicatii și controlul bordului)
- **industria medicală** (dezvoltarea aparaturii medicale, garantând corectitudinea algoritmilor care receptioneaza si evalueaza diferite semnale)
- **în criptografie** (la dezvoltarea algoritmilor care asigură condifențialitatea și integritatea datelor)
- **securitate cibernetică** (firewall-uri sau sisteme de detectare a intruziunilor, ar putea fi verificate în mod oficial pentru a se asigura că acestea identifică și răspund corect la amenințările de securitate fără a introduce vulnerabilități)
- **industria automobilelor autonome** (Dafny ar putea fi utilizat pentru a verifica corectitudinea algoritmilor de percepție, de luare a deciziilor și de control, reducând astfel riscul de accidente).

Capitolul 2

Programarea dinamică

Programarea dinamică este o tehnică de proiectare a algoritmilor utilizată pentru rezolvarea problemelor de optimizare. Pentru a rezolva o anumită problemă folosind programarea dinamică, trebuie să identifiăm în mod convenabil mai multe subprobleme. După ce alegem subproblemele, trebuie să stabilim cum se poate calcula soluția unei subprobleme în funcție de alte subprobleme. Principala idee din spatele programării dinamice constă în stocarea rezultatelor subproblemele pentru a evita recalcularea lor de fiecare dată când sunt necesare. În general, programarea dinamică se aplică pentru probleme de optimizare pentru care algoritmii greedy nu produc în general soluția optimă.

2.1 Probleme rezolvate cu ajutorul programării dinamice

Urmatoarele probleme pot fi rezolvate cu ajutorul programarii dinamice:

- **Problema șirului lui Fibonacci**
- **Problema buturugii**
- **Problema distanței de editare**
- **Problema plătii unei sume de bani folosind numar minim de bancnote**
- **Problema înmultirii optime a unui sir de matrici**
- **Problema de selecție a activităților cu profit maxim**

2.1.1 Problema de selecție a activităților cu profit maxim

Problema de selecție a activităților cu profit maxim este o problema de optimizare care returneaza pentru o listă de activități caracterizate prin timp de început, timp de sfârsit și profit, ordonate dupa timpul de sfârșit o secventă de activități care nu se suprapun și al caror profit este maxim.

```
Input: Numarul de activități n = 4
Detaliile activităților { timp de inceput, timp de incheiere,  prof
   Activitate 1:  {1, 2, 50}
   Activitate 2:  {3, 5, 20}
   Activitate 3:  {6, 19, 100}
   Activitate 4:  {2, 100, 200}
Output: Profit-ul maxim este 250, pentru soluția optimă formata din
activitatea 1 si activitatea 4.
```

2.1.2 Pseudocod

```
struct Activitate {
    timp de început: int
    timp de încheiere : int
    profit : int
}

function planificareActivitățiPonderate(activități):
    // activitățile sunt sortate crescător după timpul de încheiere

    // inițializăm un vector pentru a stoca profiturile maxime pentru f
    dp[0] = activități[0].profit
    solutie[0] = [activitati[0]] //un vector binar 0 - nu am ales activ
    solutiiOptime = solutie //stocam soluțiile optime la fiecare pas
    // Programare dinamică pentru a găsi profitul maxim
    pentru i de la 1 la n-1:
        // Găsește cea mai recentă activitate care nu intră în conflict
        solutiaCuActivitateaI = [1] // selectam activitatea curenta
        ultimaActivitateNeconflictuală = găsesteUltimaActivitateNeconfl
```

```
        // Calculează profitul maxim incluzând activitatea curentă și e
        profitulCuActivitateaCurenta = activități[i].profit
        dacă ultimaActivitateNeconflictuală != -1:
            profitulIncluderiiActivitățiiCurente += dp[ultimaActivitate
            solutiaCuActivitateaI = solutiiOptime[ultimaActivitateNecon
        dp[i] = maxim(profitulIncluderiiActivitățiiCurente, dp[i-1])

        daca profitulIncluderiiActivitățiiCurente > dp[i-1]:
            solutie = solutieCuActivitateaI
        else
            solutie = solutie + [0]
        solutiiOptime = solutiiOptime + solutie
    // Returnează soluția si profitul maxim
    return solutie, dp[n-1]

funcție găsețteUltimaActivitateNeconflictuală(activități, indexCurent):
    pentru j de la indexCurent-1 la 0:
        dacă activități[j].timpDeÎncheiere <= activități[indexCurent].t
            return j
    return -1
```

2.1.3 Cum funcționeaza programarea dinamică în cazul acestei probleme

Pentru aceasta problemă se poate folosi programarea dinamică, deoarece aceasta poate fi împarțită în subprobleme, respectiv la fiecare pas putem forma o soluție parțiala optimă, de al carei profit ne putem folosi la urmatorul pas. La fiecare pas trebuie să selectăm o activitate în ordinea în care au fost declarate în secvența de intrare (formăm soluția parțială ce conține activitatea curentă), să cautam dacă există în fața acestora activități care nu se suprapun cu ele, daca da, concatenăm cu soluția partiala optimă

formată cu activități de pâna la activitatea cu care nu se suprapune, apoi dacă aceasta soluție partiala ce conține activitatea curentă are un profit strict mai mare decat cel optim anterior, o alegem, în caz contrar, alegam să nu selectam această activitate. La primul pas soluția parțiala optimă de lungime 1 este formata din Activitatea 1, iar profit-ul maxim este 50. La al 2-lea pas, deoarece activitatea 1 nu se suprapune cu activitatea 2 se obține soluția partiala optimă formata din Activitatea1 și Activitatea2, iar profitul optim la pasul 2 este 50 + 20 = 70, care este mai mare decat cel anterior (condiție necesară). La al 3-lea pas soluția parțială optimă este formata din Activitatea 1, Activitatea 2 și Activitatea 3, deoarece Activitatea 3 nu se suprapune cu activitatea 2 (înseamnă că nu se suprapune cu soluția partiala de la al 2-lea pas, fiind ordonate dupa timpul se sfârsit) și facem o concatenare cu soluția parțiala de la pas-ul 2 , și totodata profitul pentru această soluție parțiala este strict mai mare decat cel de la pasul 2, noul profit optim devenind 170. La al 4-lea pas, la fel ca și la ceilalți formăm mai întâi o soluție partială care continea Activitatea 4. Selectăm Activitatea 4 și parcurgem secvența de activități (de la actvitatea 4 catre activitatea 1) data ca input pentru a putea găsi dacă există o activitate cu care aceasta să nu se suprapună. Singura activitate cu care nu se suprapune este activitatea 1, și soluția parțială ce conține Activitatea 4 este formata din Activitatea 4 si Activitatea 1. Apoi verificam daca profitul pentru aceasta soluție parțiala = 250 este strict mai mare decat profitul optim anterior = 170, adevărat. Acest lucru înseamnă că soluția parțiala optimă de lungime 4 care este și soluția problemei este formata din Activitatea 1 și Activitatea 4.

2.2 Avantajele programării dinamice față de celelalte tehnici de proiectare

În ceea ce privește programarea dinamică și tehnica greedy, în ambele cazuri apare noțiunea de subproblemă și proprietatea de substructură optimă. De fapt, tehnica greedy poate fi gândită ca un caz particular de programare dinamică, unde rezolvarea unei probleme este determinată direct de alegerea greedy, nefiind nevoie de a enumera toate alegerile posibile. Avantajele programării dinamice fața de tehnica greedy sunt:

- **Optimizare Globală**: : Programarea dinamică are capacitatea de a găsi soluția optimă globală pentru o problemă, în timp ce algoritmii greedy pot fi limitați la

luarea deciziilor locale care pot duce la o soluție suboptimală.

- **Flexibilitate**: Programarea dinamică poate fi utilizată pentru o gamă mai largă de probleme, inclusiv cele care implică restricții mai complexe sau soluții care necesită evaluarea mai multor posibilități. În comparație, algoritmii greedy sunt adesea limitați la problemele care pot fi rezolvate prin luarea deciziilor locale în fiecare pas.

Cu toate acestea, algoritmii greedy pot fi mai potriviți pentru problemele care permit luarea de decizii locale și producerea rapidă a unei soluții aproximative.

Capitolul 3

Problema de selecție a activităților cu profit maxim

Amet venenatis urna cursus eget. Quam vulputate dignissim suspendisse in est ante. Proin nibh nisl condimentum id. Egestas maecenas pharetra convallis posuere morbi. Risus viverra adipiscing at in. Vulputate eu scelerisque felis imperdiet. Cras adipiscing enim eu turpis egestas pretium aenean pharetra. In aliquam sem fringilla ut morbi tincidunt augue. Montes nascetur ridiculus mus mauris. Viverra accumsan in nisl nisi scelerisque eu ultrices vitae. In nibh mauris cursus mattis molestie a iaculis. Interdum consectetur libero id faucibus nisl tincidunt eget. Gravida in fermentum et sollicitudin ac orci. Suscipit adipiscing bibendum est ultricies. Etiam non quam lacus suspendisse. Leo urna molestie at elementum eu facilisis sed odio morbi. Egestas congue quisque egestas diam in arcu cursus. Amet consectetur adipiscing elit ut aliquam purus.

3.1 Descrierea algoritmului

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut

lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer feugiat scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

3.2 Datele problemei

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volutpat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

3.2.1 Reprezentarea datelor de intrare si a celor de ieșire

3.2.2 Tipuri de date folosite în dezvoltarea problemei

3.3 Implementarea propriu-zisă

3.3.1 Detalii de implementare

3.3.2 Metode, funcții, lemme si predicate folosite

3.3.3 Precondiții, postcondițiilor și invarianți

3.3.4 Timeout

3.4 Mod de lucru

3.4.1 Assume false

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

- Author1, *Book1*, 2018
- Author2, *Boook2*, 2017
- https://cgi.cse.unsw.edu.au/ eptcs/paper.cgi?FROM2019.1.pdf
- https://dafny.org/dafny/toc
- https://www.geeksforgeeks.org/weighted-job-scheduling/
- https://sites.google.com/view/fii-pa/2022/lectures?authuser=0
- https://profs.info.uaic.ro/ stefan.ciobaca/wollic2021slides.pdf
- https://drive.google.com/file/d/1jybqXbYpFlch54SSPnJSqC6Xxdb4bibF/view