МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Методи та системи штучного інтелекту

Комп'ютерний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Системи і методи штучного інтелекту»
спеціальності 122 Комп'ютерні науки

Укладачі: Шаповал Н.В.

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент Зайченко О.Ю., д.т.н., професор, кафедра математичних методів системного аналізу НН ІПСА

Відповідальний редактор Джигирей І.М., к.т.н., доцент кафедри штучного інтелекту ННІПСА

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 4 від 19.01.2023 р.) за поданням Вченої ради навчально-наукового інституту прикладного системного аналізу (протокол № 4 від 19.12.2022 р.)

У навчальному посібнику викладено основний теоретичний матеріал до відповідних практичних завдань щодо класичних задач штучного інтелекту: пошук в просторі станів, задачі планування, пошук дій в іграх тощо. Містяться методичні вказівки щодо виконання практичних завдань, матеріал для підготовки до занять під час самостійної роботи. При складанні посібника автор ставив за мету викласти предмет просто та наочно.

Посібник призначений для бакалаврів, які навчаються за спеціальністю 122 «Комп'ютерні науки», а також для всіх, хто використовує методи штучного інтелекту у своїй практичній діяльності.

Реєстр. № НП 22/23‑363. Обсяг 0,9 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

Вступ

Штучний інтелект (ШІ) займається не тільки розумінням, а й створенням інтелектуальних сутностей — машин, які можуть обчислювати, діяти ефективно та безпечно в найрізноманітніших нових ситуаціях. ШІ одна з найцікавіших і найбільш швидкозростаючих галузей. В даний час ШІ охоплює величезну різноманітність підсфер, починаючи від загальних (навчання, міркування, сприйняття тощо) до конкретних, наприклад, гра в шахи, доведення математичних теорем, написання віршів, водіння автомобіля чи діагностика захворювань. ШІ актуальний для будь-якого інтелектуального завдання.

Підхід до побудови ШІ на основі раціональних агентів є найбільш загальним. Для розв’язку задач планування, складання розкладів, розпізнавання, класифікації тощо необхідне створення агента, який на основі знання про світ та про правила взаємодії зі світом можете виконувати дії, які і приведуть його до поставленої мети.

На початкових етапах розвитку ШІ дослідники намагалися створити універсального агента для розв’язку широкого кола задач - і не досягли успіху. Сучасні дослідники концентруються на окремих областях ШІ та часто створюють системи, які поєднують різні напрямки ШІ.

Комп’ютерний практикум №1.

Інтелектуальні агенти

Хід виконання роботи

1. Згідно варіанту зробити опис проблемного середовища PEAS.
2. Класифікувати проблемне середовище інтелектуального агента за типом.
3. Розглянути кожний тип агента і визначити чи підходить він для вирішення поточної задачі.

Короткі теоретичні відомості

Раціональним агентом називається агент, який діє таким чином, щоб можна було досягти найкращого результату або, в умовах невизначеності, найкращого очікуваного результату. Агент сприймає навколишнє середовище за допомогою датчиків і виконує дії, які в свою чергу впливають на навколишнє середовище за допомогою виконавчих механізмів.

Агент під час свого функціонування отримає сприйняття за допомогою датчиків і на основі актів сприйняття та вбудованих знань обирає, яку дію слід виконати.

Агент служить “рішенням” певної конкретної задачі. І перш ніж проектувати агента - визначати вбудовані знання, можливі дії, тощо необхідно описати проблемне середовище в якому діє агент. Опис проблемного середовища можна виконати описавши наступні чотири пункти - продуктивність, середовище, виконавчі механізми, датчики (Performance, Environment, Actuators, Sensors) скорочено PEAS.

Приклад 1. Побудувати опис середовища для агента - автоматизований водій таксі.

	Performance	Environment	Actuators	Sensors
Водій таксі	Швидка та	Лівосторонні	Рульове	Камера,

	комфортна поїздка, дотримання ПДР, максимізувати вигоду, мінімізувати вплив інших учасників дорожнього руху	й чи правостороннній рух, можуть бути опади, ожеледиця, грунтова дорога, порушники ПДР, поліція, пасажири	управління, педаль газу, тормоз, сигнали, гудок, дисплей, динамік	радар, спідометр, GPS, датчики двигуна, акселеромет, мікрофони, тачскрін

Середовища в яких діють агенти умовно можна класифікувати наступним чином:

1. Повністю спостережне або частково спостережне середовище. Якщо датчики агента надають йому доступ до повної інформації про стан середовища в кожен момент часу, то таке проблемне середовище називається таким, що повністю спостерігається.

2. Детерміноване або стохастичне середовище. Якщо наступний стан середовища повністю визначається поточним станом і дією, виконаною агентом, то таке середовище називається детермінованим; в іншому випадку воно є стохастичним.

3. Послідовне або епізодичне середовище. В епізодичному проблемному середовищі досвід агента складається з нерозривних епізодів. Кожен епізод включає в себе сприйняття середовища агентом, а потім виконання однієї дії. В послідовних варіантах середовища поточне рішення може вплинути на всі майбутні рішення.

4. Статичне або динамічне середовище. Якщо середовище може змінитися в ході того, як агент вибирає чергову дію, то таке середовище називається динамічним для даного агента; в іншому випадку воно є статичним. Якщо з плином часу саме середовище не змінюється, а змінюються показники продуктивності агента, то таке середовище називається напівдинамічним.

5. Дискретне або безперервне середовище. Різниця між дискретними і безперервними варіантами середовища може відноситися до стану середовища,

способу обліку часу, а також сприйняття і дій агента.

6. Одноагентне або мультиагентне середовище.

7. Відоме або невідоме середовище.

Приклад 2. Для автоматизованого таксі та для гри в шахи з годинником описати тип середовища в якому діє агент

	Повністю спостережне або частково спостережне	Детермінований або стохастичне	Послідовне або епізодичне	Статичне або динамічне	Дискретне або безперервне	Одноагентне або мультиагентне
Водій таксі	частково	стохастичне	послідовне	динамічне	безперервне	мультиагентне
Гра в шахи з годинником	повністю	детермінований	послідовне	напівдинамічне	дискретне	мультиагентне

Існують наступні типи агентів:

- Простий рефлекний агент. Подібні агенти вибирають дії на основі поточного акту сприйняття, ігноруючи всю решту історію актів сприйняття;

- Агент, заснований на моделі. В цих агентах підтримується внутрішній стан, який відображає деякі з неспостережних аспектів навколишнього світу (тобто є “правила за якими існує світ”);

- Агент, заснований на цілі. Такі агенти мають не тільки опис поточного стану, а й свого роду інформація про цілі , які описують бажані ситуації;

- Агент, заснований на корисності. Такий агент для кожного стану світу, може бути оцінений за корисністю для агента;

- Агент, що навчається. Агент задний навчатися, що дозволяє агенту функціонувати в спочатку невідомих йому варіантах середовища і ставати більш компетентним у порівнянні з тим, що могли б дозволити тільки його початкові знання.

Задачі та вправи

1. Дати опис середовища для наступних агентів: робот-дослідник океанів титану, гра в футбол, гра в теніс, в’язання светра, медична діагностична система
2. Дати опис типу середовища для агентів зі вправи 1.
3. Навести приклади для кожного типу агенту
4. Дати відповідь на питання: чи правда що для раціонального агента потрібно знати всі наслідки власних дій?
5. Подумайте навіщо потрібен опис середовища PEAS?

Комп’ютерний практикум №2.

Неінформований пошук

Хід виконання роботи

1. Згідно варіанту представити задачу, як задачу пошуку та зробити відповідний опис.
2. Задати граф пошуку згідно пункту 1.
3. Реалізувати пошук в ширину, пошук в глибину, пошук з ітеративним поглибленням, пошук з обмеженням глибини та двонаправлений пошук на мові Python.
4. Застосувати алгоритми неінформованого пошуку до поставленої задачі (пункт 2). Порахувати кількість відвіданих вузлів, та час роботи алгоритму.
5. Порівняти алгоритми, результати звести в таблицю.

Короткі теоретичні відомості

Для того щоб визначити задачу як задачу пошуку необхідно визначити:

1. Множину всіх можливих станів в яких може бути середовище, яка називається простором станів S.

2. Початковий стан з множини станів S, в якому агент приступає до роботи, $s_i \in S$

4. Множину дій доступних для агента. Actions(s) = набір дій, які можна виконати в стані s, а є однією з допустимих дій в стані s.

5. Модель переходу, яка описує що кожна дія робить. RESULT(s, a) повертає стан, що виникає в результаті виконання дії a в стані s.

3. Визначити цільовий стан та визначити як можна зробити перевірку мети, що дозволить визначити, чи є даний конкретний стан цільовим станом. IsGoal(s) повертає чи s цільовий стан.

Шляхом в просторі станів є послідовність станів, з'єднаних послідовністю дій.

Рішенням задачі є шлях від початкового стану до цільового стану.

Оптимальним рішенням є таке рішення, яке має найменшу вартість шляху серед всіх інших рішень.

Рішення задачі досягається за допомогою пошуку в просторі станів. Пошук здійснюється за допомогою явно заданого дерева пошуку, що створюється за допомогою початкового стану і функції RESULT(s, a), які спільно задають простір станів.

Дерево пошуку формується поступово в процесі пошуку. Колекція вузлів дерева, які були сформовані, але ще не розгорнуті називається периферією. Кожен елемент периферії являє собою листовий вузол, тобто вузол, який не має наступників в дереві.

При оцінці алгоритму оцінюють повноту, оптимальність, часову та просторову складність. Повнота говорить чи гарантує алгоритм виявлення рішення, якщо воно є. Оптимальність - чи забезпечує алгоритм знаходження оптимального рішення. Якщо дерево пошуку представлено неявно за допомогою початкового стану і функції яка виначає наступний стан, і часто є нескінченним, то складність виражається в термінах трьох величин: b- коефіцієнт розгалуження або максимальну кількість наступників будь-якого вузла, d- глибина самого поверхневого цільового вузла і m- максимальна довжина будь-якого шляху в просторі станів.

Часова складність часто вимірюється в термінах кількості вузлів, що виробляються в процесі пошуку, а просторова складність в термінах максимальної кількості вузлів, що зберігаються в пам'яті.

Пошук в ширину це проста стратегія, в якій спочатку розгортається кореневий вузол, потім всі наступники кореневого вузла, після цього розгортаються наступники цих наступників і т.д. При пошуку в ширину, перш ніж відбувається розгортання будь-яких вузлів на наступному рівні, розгортаються всі вузли на даній конкретній глибині в дереві пошуку. В пошуці в ширину периферія задається за допомогою черги FIFO.

Пошук в глибину завжди розгортає найглибший вузол в поточній периферії дерева пошуку. Пошук безпосередньо переходить на найглибший рівень дерева пошуку, на якому вузли не мають спадкоємців. У міру того як ці вузли розгортаються, вони видаляються з периферії, тому в подальшому пошук "поновлюється" з наступного самого поверхневого вузла, який все ще має недосліджених наступників (дочірніх вузлів). Використовує чергу LIFO.

При пошуку з обмеженням глибини під час пошуку заздалегідь визначають межі глибини А. Це означає, що вузли на глибині А розглядаються таким чином, як якщо б вони не мали спадкоємців.

Пошук з ітеративним поглибленням дозволяє знайти найкращу межу глибини. Це досягається шляхом поступового збільшення межі (яка спочатку стає рівною 0, потім 1, потім 2 і т.д.) до тих пір, поки не буде знайдена мета (цільовий стан). Така подія відбувається після того, як межа глибини досягає значення d, глибини самого поверхневого цільового вузла.

В основі двонаправленого пошуку лежить така ідея, що можна одночасно проводити два пошуки (в прямому напрямку, від початкового стану, і в зворотному напрямку, від мети), зупиняючись після того, як два процеси пошуку зустрінуться на середині.

Приклад 1. Нехай у нас є робот, який доставляє посилки. Є початкова позиція і є цільова (куди треба доставити посилку). Карта світу на рис.1. Жирним позначено перепони, які робот не може перетинати. Робот може

рухатися вперед, назад, вліво, вправо. За межі карти він не може вийти. Необхідно побудувати дерево пошуку, та знайти рішення за допомогою, не інформованих алгоритмів пошуку.

Стан світу - те де саме буде перебувати робот задамо за допомогою системи координат. Стартова позиція буде 1а, цільова 1в.

	а	б	в
1	старт		ціль
2			
3			

Рис. 1

Побудуємо дерево пошуку Рис.2.

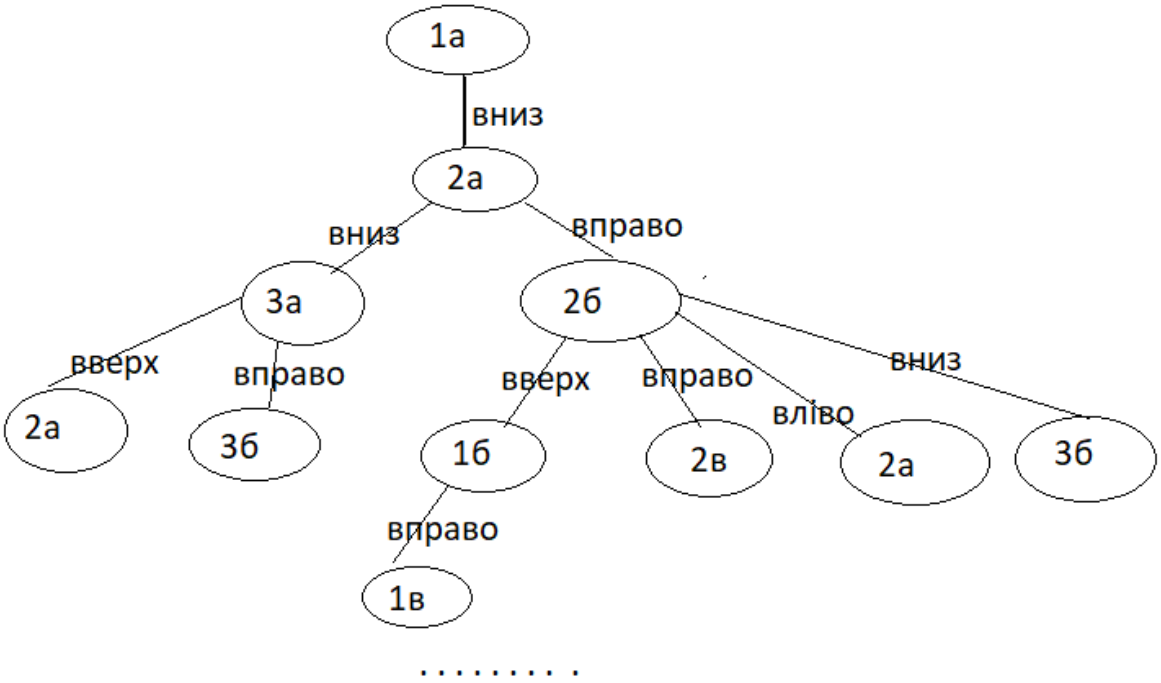


Рис.2

Дерево розгортається нескінченно далі, оскільки робот може потрапляти в вже відвідані стани. Дія, які застосовуємо в певному стані підписані на ребрах.

Пошук в ширину.

1. Розгортаємо вузол 1а, периферія (черга) [2а]. Вузол 2а вже сформований, але ще не розгорнутий. Обираємо перший вузол з периферії і перевіряємо чи він цільовий, якщо ні то розгортаємо його.

2. Розгортаємо вузол 2а, периферія [3а, 2б]. Вибираємо перший вузол з черги - 3а. Він не цільовий.

3. Розгортаємо вузол 3а, периферія [2б, 2а, 3б]. Вибираємо перший вузол з черги - 2б. Він не цільовий.

4. Розгортаємо вузол 2б, периферія [2а, 3б, 1б, 2в, 2а, 3б]. Вибираємо перший вузол з черги - 2а. Він не цільовий.

5. Розгортаємо вузол 2а, периферія [3б, 1б, 2в, 2а, 3б, 3а, 3б]. Вибираємо перший вузол з черги - 3б. Він не цільовий.

6. Розгортаємо вузол 3б, периферія [1б, 2в, 2а, 3б, 3а, 3б, 2б, 3в, 3а]. Вибираємо перший вузол з черги - 1б. Він не цільовий.

7. Розгортаємо вузол 1б, периферія [2в, 2а, 3б, 3а, 3б, 2б, 3в, 3а, 1в, 2б]. Вибираємо перший вузол з черги - 2в. Він не цільовий.

І так далі.

В периферії вже з'явився цільовий вузол, але ми маємо розгорнути всі вузли на рівні чотири перш ніж перейти на рівень п’ять, де знаходиться цільовий вузол.

Можемо відстежувати повторювані стани і не додавати їх в чергу.

Пошук в глибину.

1. Розгортаємо вузол 1а, периферія (черга) [2а]. Вузол 2а вже сформований, але ще не розгорнутий. Обираємо перший вузол з периферії і перевіряємо чи він цільовий, якщо ні то розгортаємо його.

2. Розгортаємо вузол 2а, периферія [3а, 2б]. Вибираємо перший вузол з черги - 3а. Він не цільовий.

3. Розгортаємо вузол 3а, периферія [2а, 3б, 2б]. Вибираємо перший вузол з черги - 2а. Він не цільовий.

4. Розгортаємо вузол 2а, периферія [3а, 2б, 3б, 2б]. Вибираємо перший вузол з черги - 3а. Він не цільовий.

5. Розгортаємо вузол 3а, периферія [2а, 3б, 3а, 2б, 3б, 2б]. Вибираємо перший вузол з черги - 2а. Він не цільовий.

Отже пошук пішов по нескінченному шляху рішення.

Двонаправлений пошук.

Застосуємо пошук в ширину в обидва боки.

В прямому напрямку

1. Розгортаємо вузол 1а, периферія [2а].

В зворотньому напрямку

1. Беремо вузол 1в, перевіряємо чи є він на периферії пошуку в прямому напрямку. Ні. Розгортаємо вузол 1в, периферія [1б, 2в].

В прямому напрямку

2. Беремо вузол 2а, перевіряємо чи є він на периферії пошуку в прямому напрямку. Ні. Розгортаємо вузол 2а, периферія [3а, 2б].

В зворотньому напрямку

2. Беремо вузол 1б, перевіряємо чи є він на периферії пошуку в прямому напрямку. Ні. Розгортаємо вузол 1б, периферія [2в, 2б, 1в].

В прямому напрямку

3. Беремо вузол 3а, перевіряємо чи є він на периферії пошуку в прямому напрямку. Ні. Розгортаємо вузол 3а, периферія [2б, 3б, 2а].

В зворотньому напрямку

3. Беремо вузол 2в, перевіряємо чи є він на периферії пошуку в прямому напрямку. Ні. Розгортаємо вузол 2в, периферія [2б, 1в, 2б, 1в].

В прямому напрямку

4. Беремо вузол 2б, перевіряємо чи є він на периферії пошуку в прямому напрямку. Так. Отже ми знайшли рішення.

Дії: вниз, вправо, вправо, вверх.

Контрольні запитання

1. Що таке задача пошуку? Як задається задача пошуку?
2. Як працює пошук в ширину?
3. Як працює пошук в глибину?
4. Як працює пошук в глибину з обмеженням глибини?
5. Як працює пошук в глибину з ітеративним поглибленням?
6. Як працює двонаправлений пошук?
7. Яка обчислювальна та просторова складність вивчених алгоритмів?

Задачі та вправи

1. Виконати пошук в ширину, в глибину, з ітеративним поглибленням та двонаправлений пошук для задачі де модель переходу задана функцією F(n)=2n та F(n)=2n+1. Початковий вузол 1, цільовий 11.
2. Представити задачу як задачу пошуку. Виконати пошук в ширину, в глибину, з ітеративним поглибленням та двонаправлений пошук для наступної задачі задачі: Є два глечики 5 та 8 літрів, є кран з водою. В горщики можна наливати воду та виливати з них. Необхідно що одному з горщиків був рівно 1 літр.
3. Дайте відповідь на питання: В якому випадку двонаправлений пошук буде оптимальним і повним?
4. Дайте відповідь на питання: який з неінформованих алгоритмів пошуку краще підійде до задачі пошуку страви в інтернеті?

Комп’ютерний практикум №3.

Інформований пошук

Хід виконання роботи

1. Згідно варіанту представити задачу, як задачу пошуку та зробити відповідний опис.
2. Задати граф пошуку згідно пункту 1.
3. Реалізувати жадібний пошук, пошук A*, ітеративний пошук A* на мові Python.
4. Застосувати алгоритми інформованого пошуку до поставленої задачі (пункт 2). Порахувати кількість відвіданих вузлів, та час роботи алгоритму. Якщо задача передбачає використання різних евристик, то порівняти їх роботу.
5. Порівняти алгоритми, результати звести в таблицю.

Короткі теоретичні відомості

Інформований алгоритм пошуку - це такий алгоритм де використовується інформація про задачі, що дозволяє швидше знайти мету (цільовий стан). Таку інформацію про мету надає евристична функція, що позначається як $h(n)$ - це оцінка вартості найменш дорогого шляху від вузла n до цільового вузла. Евристичні функції - довільні функції, пов'язані з конкретною проблемою, з одним обмеженням: якщо n цільовий вузол, то $h(n)=0$.

Наприклад для пошуку маршруту з Києва до Миколаєва евристичною функцією може бути відстань по прямій від поточного вузла (міста) до Миколаєва.

В алгоритмах пошуку вибір наступного вузла для розгортання здійснюється на основі функції оцінки $f(n)$ для вузла n.

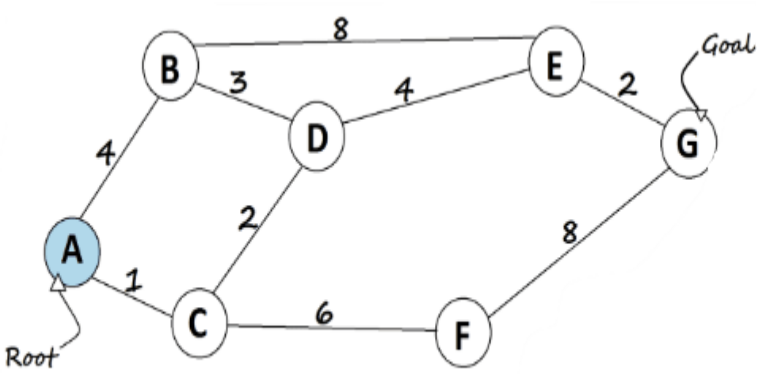
Жадібний пошук по першому найкращому збігу належить до інформованих алгоритмів пошуку. В цьому алгоритмі робляться спроби розгортання вузла, найближчого до мети на тій підставі, що він з усією

ймовірністю повинен швидко привести до рішення. Таким чином, при цьому пошуку оцінка вузлів проводиться з використанням тільки евристичної функції: $f(n)=h(n)$.

Приклад1. Нехай у нас є наступне дерево пошуку. Над ребрами що з'єднують стани вказана фактична вартість переходу між вузлами (вартість виконання дії для переході зі стану в стан). Значення евристичної функції дано в таблиці 1. Виконаємо жадібний пошук.

таблиця 1.

A	8
B	8
C	6
D	5
E	1
F	4
G	0



Для вузла C h(C)=6 для вузла B h(B)=8: A→C

Для вузла D h(D)=5 для вузла F h(F)=4: A→C →F

Для вузла G h(G)=0: A→C →F → G

A→C→F→G відстань такого маршруту 1+6+8=15

В інформованому пошуку A* відбувається мінімізація сумарної оцінки вартості рішення. В цьому алгоритмі $f(n)=h(n)+g(n)$, де $f(n)$ - оцінка вартості найменш дорогого шляху рішення, що проходить через вузол n; $h(n)$ - визначає оцінку вартості найменш дорогого шляху від вузла n до цілі; g(n) - дозволяє визначити вартість шляху від початкового вузла до вузла n.

Приклад 2. Умови із прикладу 1. Виконаємо пошук A*. Перед розгортанням вузла завжди перевіряємо чи це цільовий вузол.

1.Розгортаємо вузол S. Рахуємо для дочірніх вузлів функції оцінки f(C) =6+1, f(B)=4+8. Периферія пошуку черга з пріоритетом [AC, AB] A→C

2. Розгортаємо вузол C. Рахуємо для дочірніх вузлів функції оцінки f(D) =3+5, f(F)=7+4. Периферія пошуку черга з пріоритетом [ACD,ACF, AB] A→C →D

3. Розгортаємо вузол D. Рахуємо для дочірніх вузлів функції оцінки f(E) =7+1, f(B)=6+8. Периферія пошуку черга з пріоритетом [ACDE, ACF, AB, ~~ACDB~~ (видаляємо цей шлях, оскільки є шлях в B меншої вартості)] A→C →D →E

4. Розгортаємо вузол E. Рахуємо для дочірніх вузлів функції оцінки f(G) =9+0, f(B)=23. Периферія пошуку черга з пріоритетом [ACDEG, ACF, AB] A→C →D →E→G.

5. Цільовий вузол G.

Відстань такого маршруту 1+2+4+2=9

A* є оптимальним, за умови, що h(n) являє собою допустиму евристичну функцію, тобто за умови, що h(n) ніколи не переоцінює вартість досягнення мети. Допустимі евристичні функції повертають значення вартості рішення задачі, менші в порівнянні з фактичними значеннями вартості. А оскільки g(n)-точна вартість досягнення вузла n, отже, що функція f(n) ніколи не переоцінює справжню вартість досягнення рішення через вузол n.

Евристична функція h(n) є монотонною, якщо для будь-якого вузла n і для будь-якого n' - дочірнього вузла n, сформованого в результаті будь-якої дії a, оцінка вартості досягнення мети з вузла n не більше вартості етапу досягнення вузла n' плюс оцінка вартості досягнення мети з вузла n': $h(n) \leq c(n,a,n')+h(n')$.

A* з ітеративним поглибленням схожий на пошук в глибину з ітеративним поглибленням. В IDA* (iterative deepening A*) умовою зупинки розгортання служить f-вартість ($g+h$); на кожній ітерації цим критерієм зупинки є мінімальна f-вартість будь-якого вузла, що перевищує значення критерія зупинки, досягнуте в попередній ітерації.

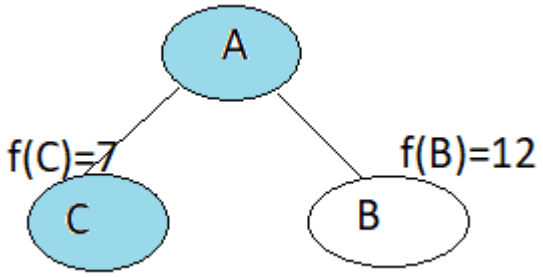
Приклад 2. Умови із прикладу 1. Виконаємо пошук IDA*.

1. На першій ітерації початкове занчення межі f-вартості дорівнює

евристичній функції в початковому вузлі.

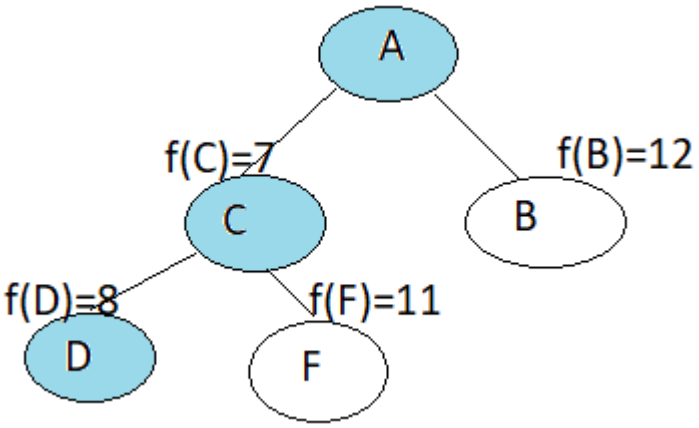
f_межа=8. Намалюємо дерево пошуку.

Розгортаємо вузол A.



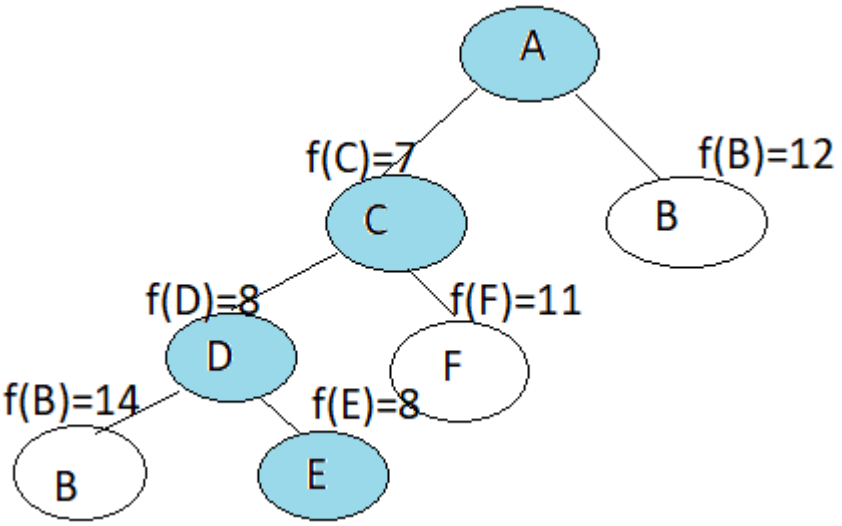
Вузол A має лише один дочірній вузол, оскільки тільки у нього функція оцінки менше значення f_межі. На позначення цього факту вузол B не зафарбований.

Розгортаємо вузол C.



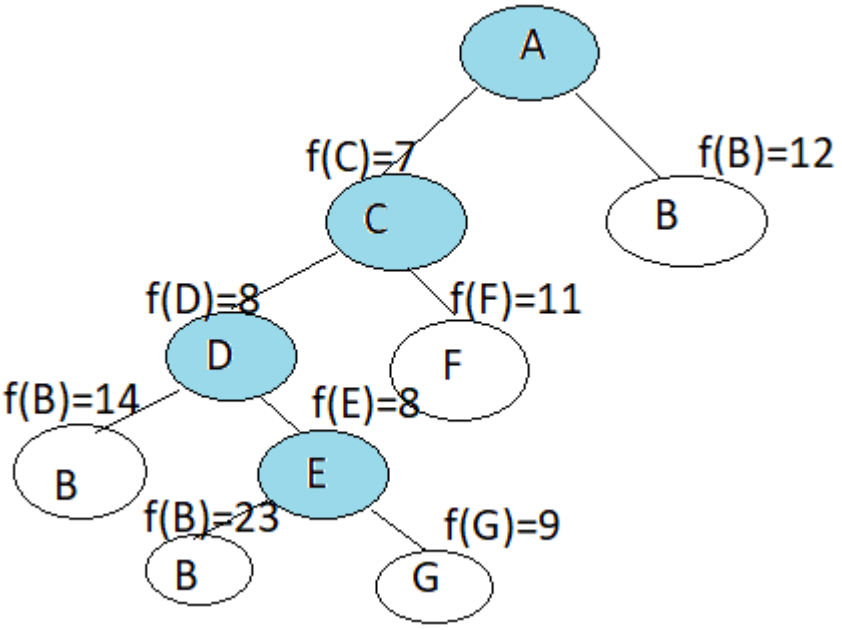
Вузол C має лише один дочірній вузол, оскільки тільки у нього функція оцінки менше або дорівнює значення f_межі.

Розгортаємо вузол D.



Вузол D має лише один дочірній вузол, оскільки тільки у нього функція оцінки менше або дорівнює значення f_межі.

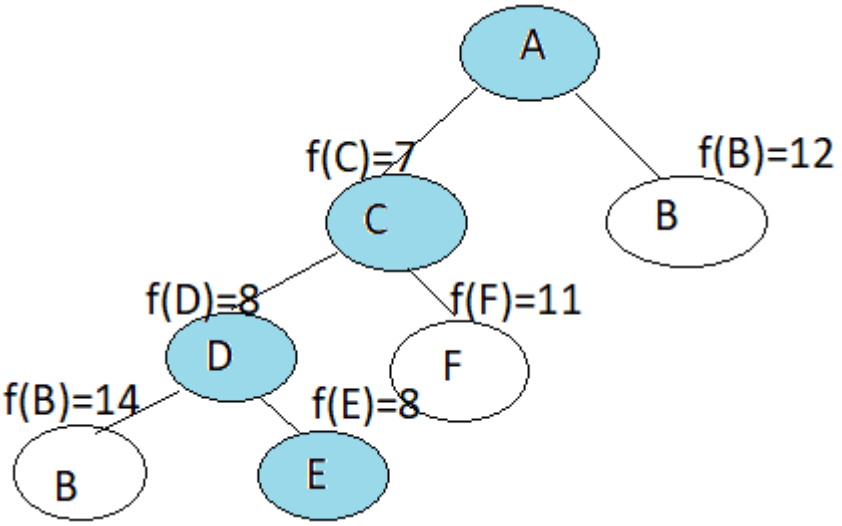
Розгортаємо вузол E.



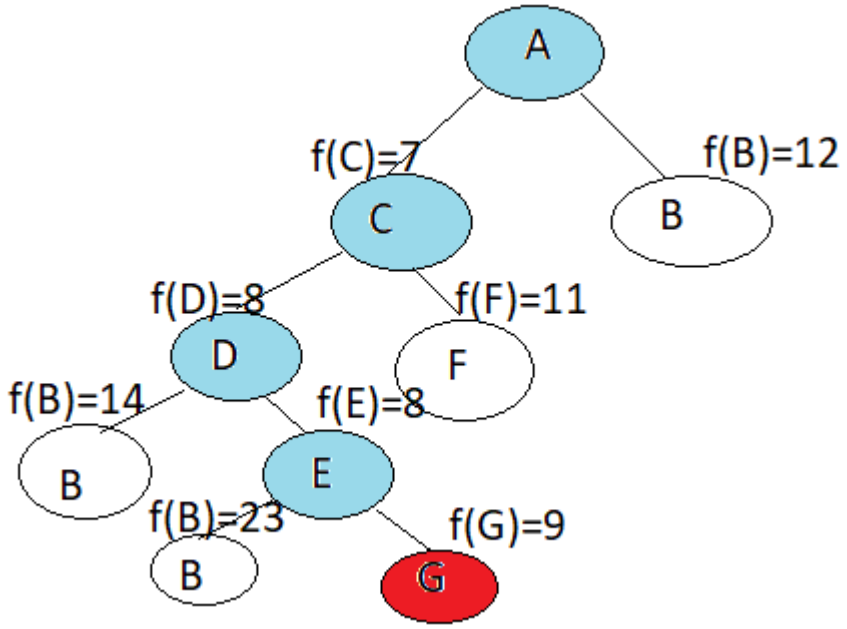
Вузол E не має дочірніх вузлів, оскільки у жодного з можливих дочірніх вузлів функція оцінки не менше або дорівнює значення f_межі. Отже необхідно визначити нову межу. Серед вузлів, що з'являлися на периферії пошуку візьмемо мінімальне значення f- вартості. Нова межа f_new=9. Починаємо наступну ітерацію.

2. f_межа=9. Повторюємо викладки які були на попередній ітерації. В

даному прикладі кроки повторюються. Перейдимо одразу до розгортання вузла E.



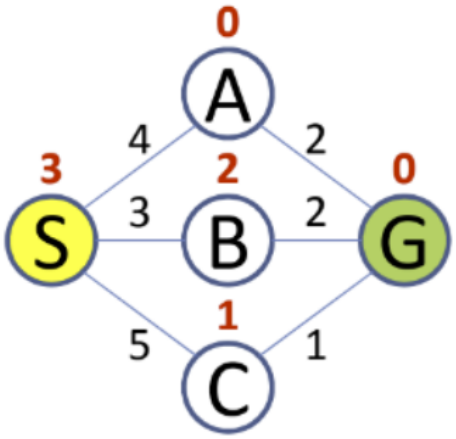
Він має лише один дочірній вузол G з оцінкою f(G)=9.



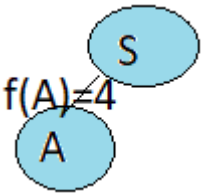
G - цільовий вузол. Маршрут A→C →D →E→G. Вартість маршруту дорівнює 9.

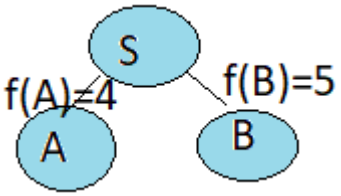
В алгоритмі A* з обмеженням пам’яті (Simplified Memory-bounded A*) відбувається обмеження на кількість вузлів, які можуть зберігатися в пам’яті. Як тільки пам’ять заповнена необхідно забути, якийсь вузол. Видяляється найгірший листовой вузол, при цьому в батьківському резервується найкраще значення функції оцінки яке було забуте, разом з вузлом що видаляється.

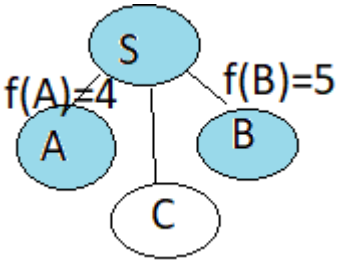
Приклад 4. Нехай в нас є граф пошуку. Початковий вузол S, цільовий G. Чорним позначені фактичні вартості шляху між вузлами, червоним позначені евристичні значення.



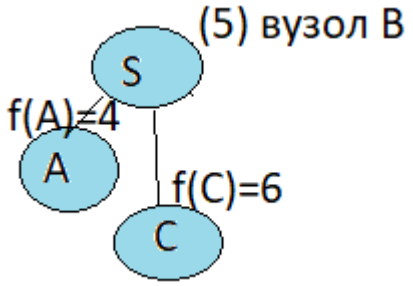
Нехай у нас є обмеження пам’яті в 3 вузли. Формуємо вузли по одному.

1. 

2. 

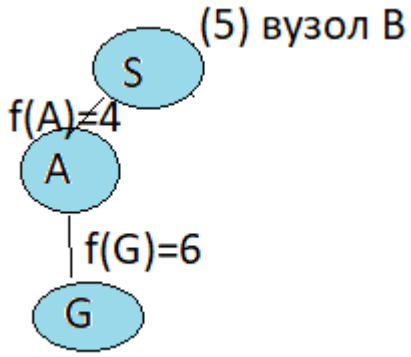
3. 

На третьому кроці пам’ять заповнена і вузол С розгорнути не можна. Необхідно забути найгірший листовий вузол. Це вузол В. Запам’ятовуємо функцію оцінки і те звідки вона.



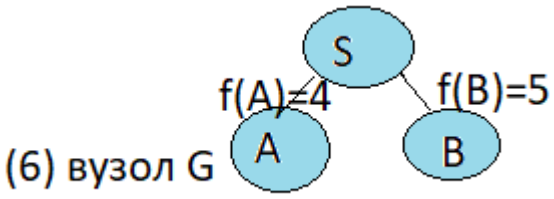
4.

На п’ятому кроці пам’ять заповнена забуваємо найгірший листовий вузол C. В батьківському вузлі ми пам’ятаємо кращей шлях, тож просто забуваємо вузол C.



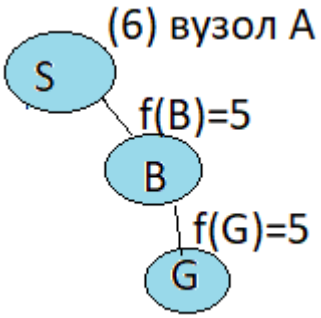
5.

Пам’ять заповнена і листовий вузол має оцінку меншу ніж ми пам’ятаємо. Тож забуваємо вузол G і запам’ятовуємо цей шлях.



6.

На шостому кроці пам’ять знову заповнена. Тож забуваємо вузол A.



7.

Тож буде знайдено шлях. Якщо б обмеження було у 2 вузли, алгоритм не

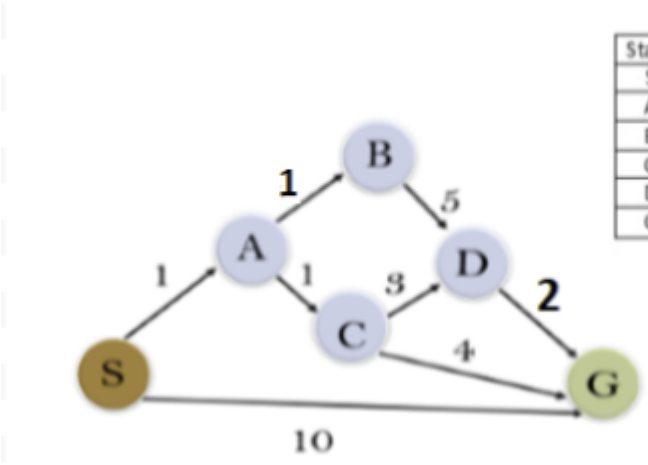
знайшов би рішення. Але повернув би частину шляху на шляху рішення.

Контрольні запитання

8. Що таке евристична функція та функція оцінки?
9. Як працює жадібний пошук?
10. Як працює пошук A*?
11. Як працює ітеративний пошук A*?
12. Як працює пошук A* з обмеженням пам’яті?
13. Яка обчислювальна та просторова складність вивчених алгоритмів?

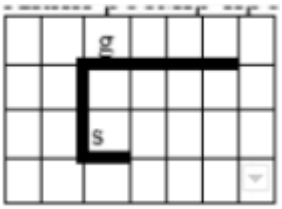
Задачі та вправи

Виконати пошук A*, IDA*, жадібний пошук, A* з обмеженням пам’яті в 3 вузли для наступного дерева пошуку:

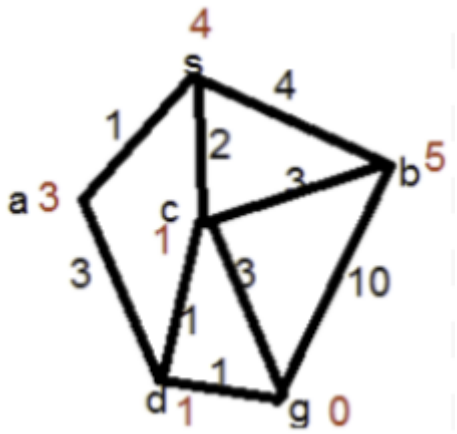
1. 

State	h1	h2
S	4	3
A	2	2
B	6	5
C	2	1
D	3	2
G	0	0

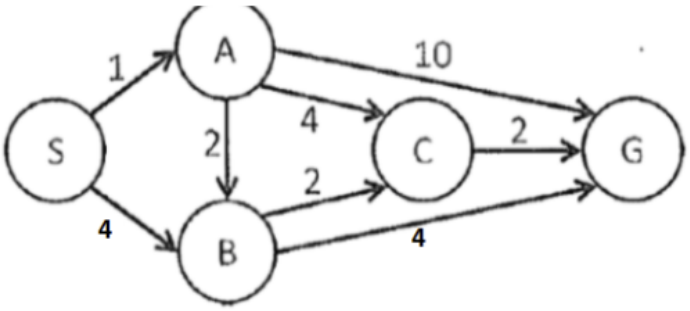
2. В якості евристики використовуйте манхетенську відстань. Початок в точці S, цільова клітинка G.



3.



4.



Node	Heuristic Estimated Distances
S	7
A	6
B	2
C	2
G	0

Комп’ютерний практикум №4. Задачі задоволення обмежень

Хід виконання роботи

1. Згідно варіанту представити задачу, як задачу задоволення обмежень та зробити відповідний опис, задати граф обмежень.
2. Використовуючи бібліотеку Simple AI (https://simpleai.readthedocs.io/en/latest/) виконати пошук з поверненням з поширенням обмежень, пошук з поверненням з перевіркою сумісності дуг та пошук з поверненням керований конфліктами для вирішення поставленої задачі.
3. Порівняти алгоритми за часом роботи.

Короткі теоретичні відомості

Задачі задоволення обмежень (CSP - Constraint Satisfaction Problem) особлива підмножина задач пошуку, яка задається:

- множиною змінних Xi;

- непрожньою областю визначення Di можливих значень для кожної змінної. Різні змінні можуть мати різні області визначення, різного розміру, типу і тд

- множиною обмежень Ci, що визначають допустимі комбінації значень для підмножин змінних (за допомогою яких відбувається перевірка цілей). Кожне обмеження можна визначити парою Ci=(Змін,обмеж), де змін визначає змінні які приймають участь в обмеженні, та обмеж - задає безпосередньо обмеження або перелік допустимих комбінацій для змінних. Наприклад, ((X1,X2),X1 > X2) або ((X1,X2), (1,2), (2,3))

Така постановка дозволяє використовувати алгоритми з більшою потужністю, ніж стандартні алгоритми пошуку, які були розглянуті до цього.

Стан в такій постановці визначається шляхом присвоєння значень деяким

або всім змінним, $\{Xi = vi, Xj = vj, \ldots\}$. Присвоєння, яка не порушує ніяких обмежень, називається сумісним присвоєнням. Повним називається таке присвоєння, в якому бере участь кожна змінна, а рішенням задачі CSP є повне присвоєння, яке задовольняє всім обмеженням.

Алгоритми пошуку CSP використовують переваги такої структури використовуючи обмеження для зменшення простору пошуку.

В термінах задач пошуку, задачу CSP можна задати як:

Початковий стан. Пусте присвоєння $\{\}$, в якому жодній змінній не присвоєно значення.

- Функція результату. Величина що може бути присвоєна будь-якій змінній з неприсвоєним значенням, за умови, що змінна не буде конфліктувати з іншими змінними, значення яких були присвоєні раніше.
- Перевірка мети. Поточне присвоєння є повним.
- Вартість шляху. Постійна вартість (наприклад, 1) для кожного етапу.

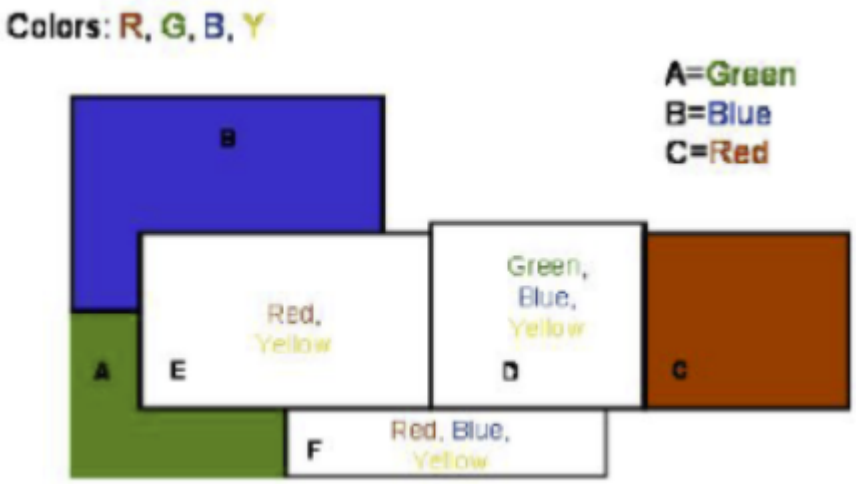
Пошук з поверненням - це Пошук в глибину, в якому кожен раз вибираються значення для однієї змінної і виконується повернення, якщо більше не залишається допустимих значень, які можна було б присвоїти змінній.

Для підвищення ефективності пошуку з поверненням можна використовувати евристики, специфічні для задач CSP.

Щоб обрати змінну для якої в подальшому необхідно зробити присвоєння можна скористатися евристикою з мінімальною кількістю решти значень (Minimum Remaining Values -MRV). Тобто для подальшого присвоєння буде обиратися змінна у якою в області визначення буде мінімальна кількість можливих значень для присвоєння.

Приклад 1. Нехай необхідно розфарбувати карту і доступні 4 кольори.

Отже 6 змінних A, B,C,D,F, для яких області визначення на початку складають всі чотири кольори. Обмеження - щоб значення змінних які відповідають сусіднім регіонам не були рівні. Нехай вже є часткове присвоєння показане на малюнку.

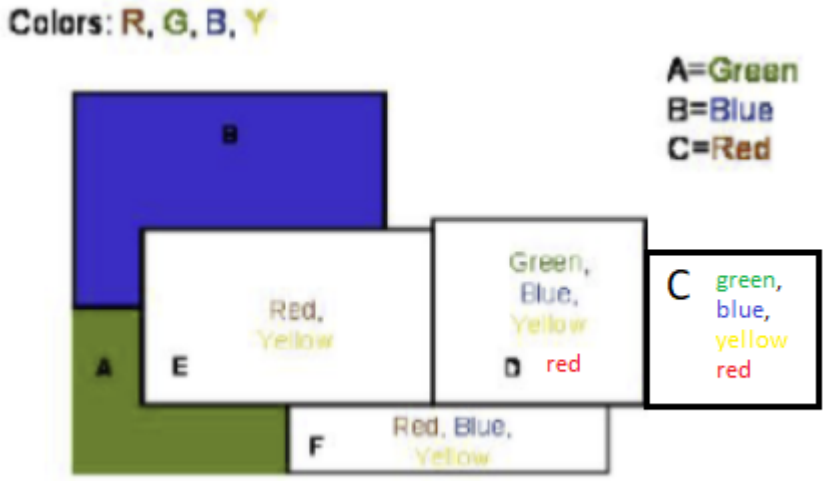


Питання: Яку змінну обрати наступною для присвоєння?

Значення евристики MRV для кожної змінної з ще не присвоєнним значенням наступні: MRV(E)=2, MRV(F)=3, MRV(D)=3. Отже за евристикою MRV для наступного присвоєння буде використовуватися змінна E.

Ступенева евристика також дає відповідь на те, яку змінну обрати наступною. У ступеневій евристиці робиться спроба зменшити коефіцієнт розгалуження в майбутніх варіантах за рахунок вибору змінної, яка бере участь в найбільшій кількості обмежень на інші змінні з неприсвоєнними значеннями.

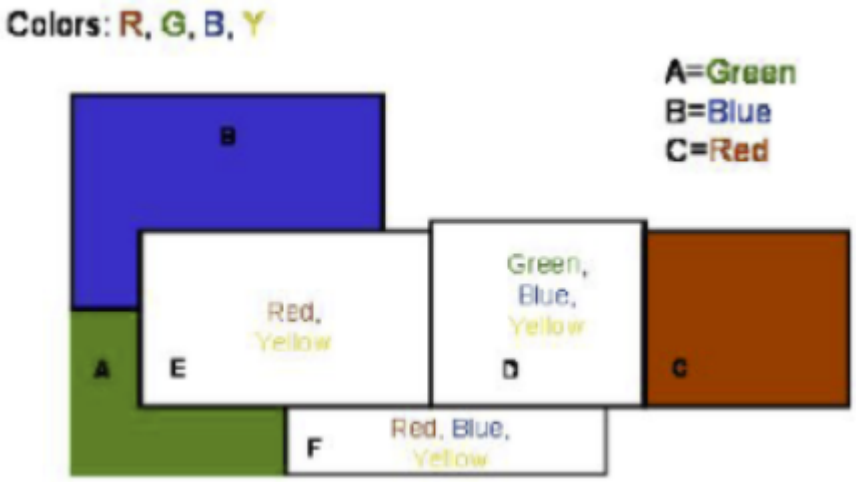
Приклад 2. Та сама задача що і в прикладі 1.



Значення ступеневої евристики для кожної змінної з ще не присвоєнним значенням наступні: Ст(E)=2, Ст(F)=2, Ст(D)=3, Ст(C)=1. Отже за ступеневою евристикою для наступного присвоєння буде використовуватися змінна D.

Щоб обрати значення, яке присвоюється змінній можна скористатися евристикою з найменш обмежувальним значенням. У ній надається перевага значенню, в якому з розгляду виключається найменша кількість варіантів вибору значень для сусідніх змінних в графі обмежень.

Приклад 3. Та сама задача що і в прикладі 1. Необхідно вибрати яке значення присвоювати змінній E.



Порахуємо евристику з найменш обмежувальним значенням для кожного

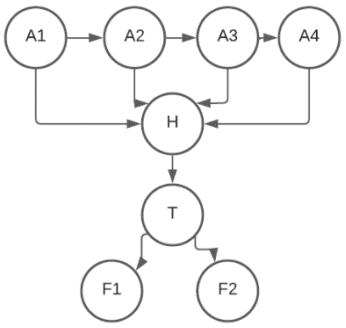
можливого присвоєння для змінної E. Для кожного варіанту порахуємо для кількох змінних це значення виключається з області значень.

НОЗ(red)=1, НОЗ(Yellow)=2. Отже оберемо значення для E Yellow.

Поширення інформації за допомогою обмежень. Попередня перевірка (forward checking) полягає в наступному: при кожному присвоєнні значення змінній X в процесі попередньої перевірки проглядається кожна змінна Y з неприсвоєним значенням, яка з'єднана з X за допомогою деякого обмеження, і з області визначення змінної Y видаляється будь-яке значення, яке є несумісним зі значенням, вибраним для X.

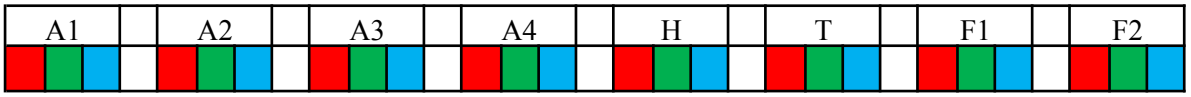
Перевірка сумісності дуг. В цьому алгоритмі поширення обмежень використовується поняття сумісності дуги: Направлена дуга що з'єднує змінні в графі обмежень x→y сумісна, якщо для кожного значення x з D1 існує значення y з D2 таке, що ці значення задовольняють обмеженню між x та y. Щоб забезпечити сумісність дуг необхідно видалити значення з області визначення D1.

Алгоритм перевірки сумісності дуг полягає в наступному: Кожна дуга (Xi , Xj) по черзі позначається як сумісна і перевіряється; якщо з області визначення Xi необхідно видалити будь-які значення, то кожна дуга (Xk , Xi), яка вказує на Xi , повинна бути повторно вставлена в чергу для перевірки.

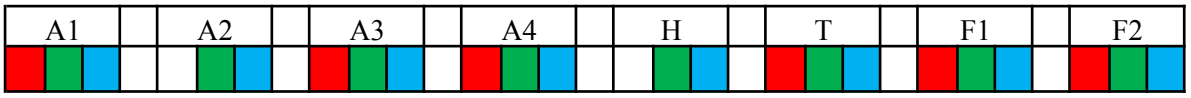


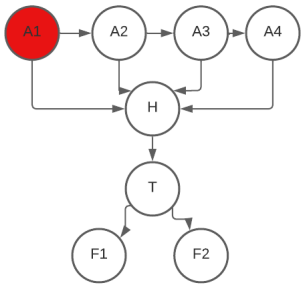
Приклад 4. Нехай є граф обмежень. Розфарбуйте граф у 3 кольори.

Розв'язок. Виконаємо пошук з поширенням обмежень використовуючи одночасно попередню перевірку та перевірку сумісності дуг. Для зручності побудуємо таблицю зі всіма змінними та областями визначення для них.



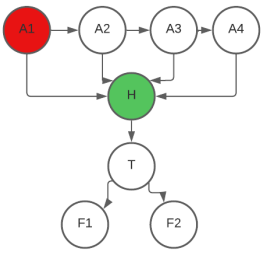
Зафарбовуємо вершину A1 в червоний колір, виконаємо попередню перевірку і видалимо червоний колір із змінних пов’язаних з A1 обмеженням і виконаємо перевірку дуг

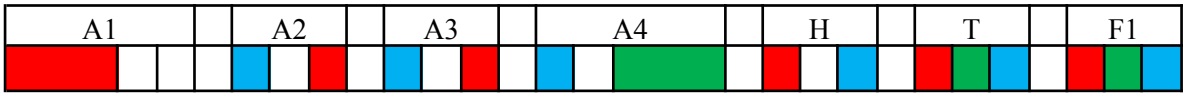




Всі дуги сумісні

Наступний вузол вибираємо за ступеневою евристикою. Це буде вузол H, який приймає участь у найбільшій кількості обмежень. Обираємо йому колір зелений і виконуємо попередню перевірку. Видалимо зелений колір із змінних пов’язаних з H обмеженням.

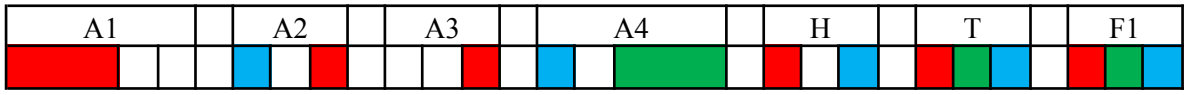




Виконаємо перевірку сумісності дугє

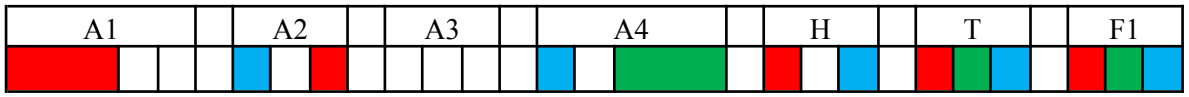
A2->A3, A3->A4, T->F2, T->F1, F1->T, F2->T – дуги сумісні.

A3->A2 несумісна, при виборі синього кольору в області значень A2 не знайдемо рішення що задовольняє обмеженню. Тож треба видалити синій колір з A3.

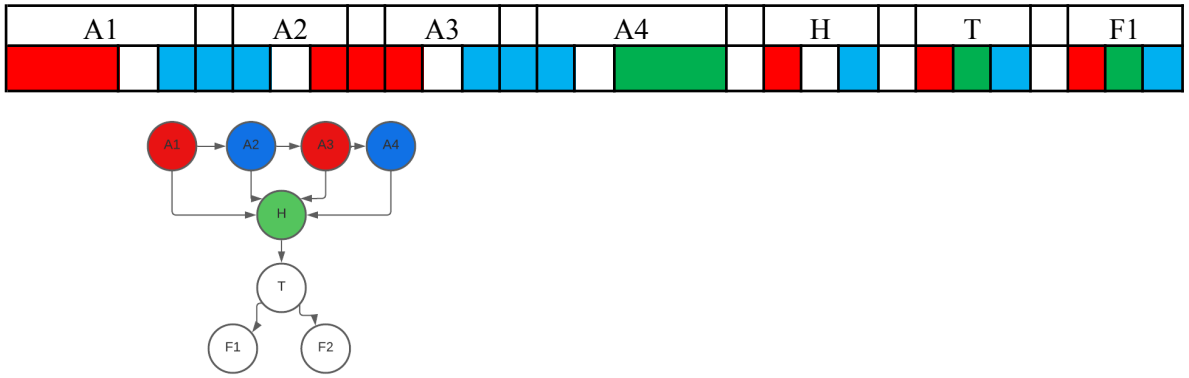


Тепер перевірити всі дуги на сумісність які вказують на A3.

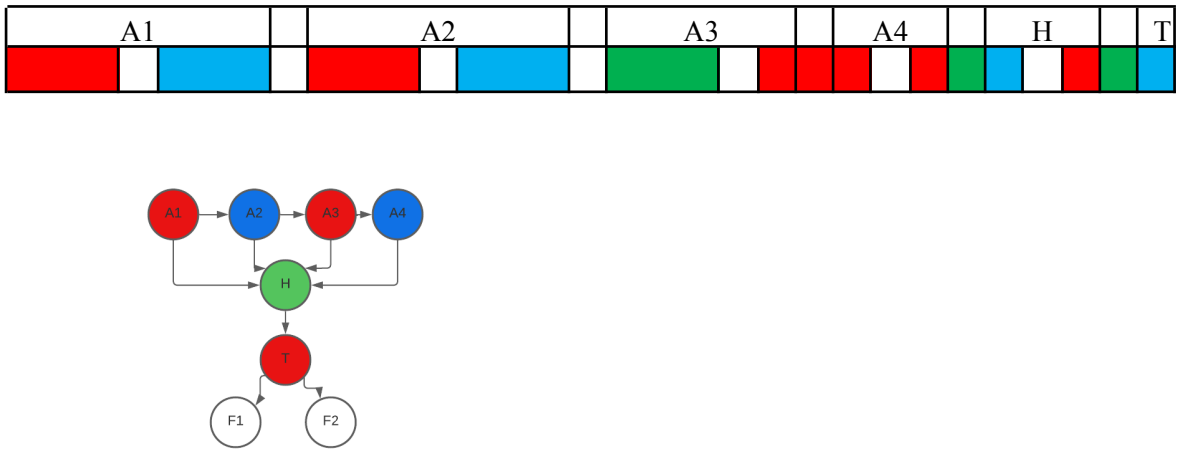
A2->A3- сумісна. A4->A3 ще не перевіряли. Дуга не сумісна. При виборі червоного кольору в A3 не знайдемо рішення що задовольняє обмеженням. Отже видаляємо червоний колір з області значень A4.



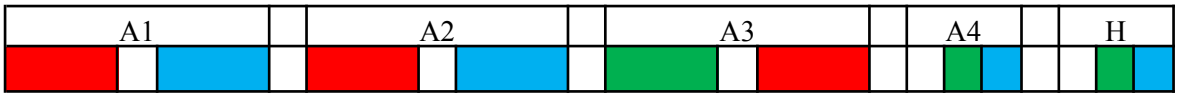
Тепер потрібно перевірити дуги які вказують на A4. Вони всі сумісні. Пропустимо частину наступних викладок бо для змінних A2, A3, A4 дуги сумісні і в області значень є лише по одному значенню. Маємо



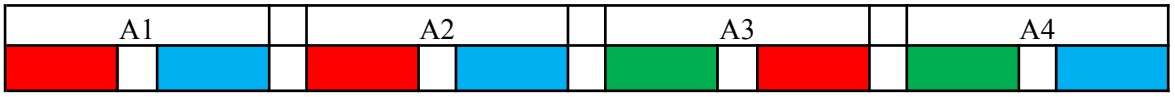
Для змінної T обираємо червоний колір.

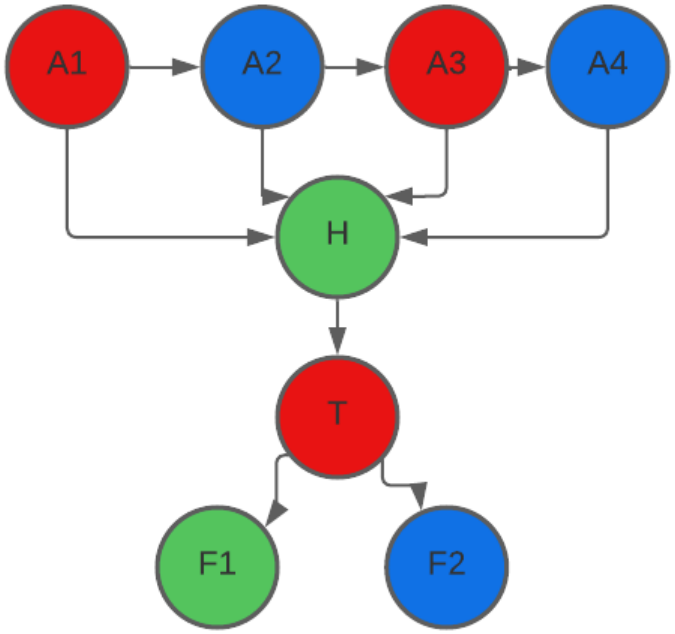


Виконуємо попередню перевірку.



Для змінних F1 та F2 встановлюємо значення і отримаємо рішення.





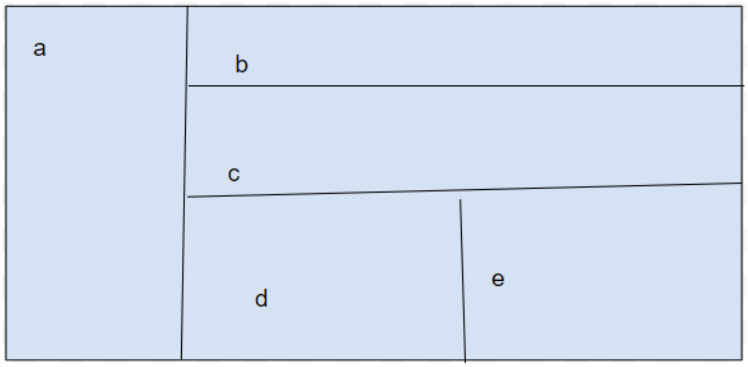
Контрольні запитання

14. Як задаються задачі задоволення обмежень?
15. Що таке пошук з поверненням?
16. Які є евристики для покращення роботи пошуку з поверненням?
17. Як працює алгоритм поширення обмежень?
18. Як працює алгоритм перевірки сумісності дуг?
19. Як працює пошук з поверненням керований конфліктами? В чому його перевага?

Задачі та вправи

Розфарбувати карту в три кольори для вправ 1-3. Виконати пошук з поверненням з поширенням обмежень, з перевіркою сумісності дуг, пошук з поверненням керований конфліктами. Показати як використовувати евристики:

1. 

2. 

3. Вершини графа V = {a, b, c, d, e, f, g} зв'язки E = {(a, g),(a, d),(b, c),(b, d),(b, g),(c, g),(d, e),(d, f),(d, g),(f, g)}.
4. Представити задачу як задачу csp. Застосувати відповідні методи рішення задач CSP (граф обмежень, пошук з поширенням обмежень, сумісність дуг усі). Весь процес рішення описати. У нас є п'ять літаків: A, B, C, D та E та дві злітно-посадкові смуги: міжнародна та внутрішня. Ми хотіли б запланувати часовий інтервал та злітно-посадкову смугу для кожного літака, щоб він або приземлився, або злетів. У нас є чотири часові інтервали: (1, 2, 3, 4) для кожної злітно-посадкової смуги, протягом яких ми можемо запланувати посадку або зліт літака. Ми повинні знайти рішення, яке відповідає наступним обмеженням: Літак B втратив двигун і повинен приземлитися в часовому інтервалі 1. Літак D може прибути в аеропорт, щоб приземлитися лише у часовий слот 3 або після нього. У літаку A закінчується пальне, і він може працювати максимум до часу 2. Літак D повинен сісти до зльоту літака C, оскільки деякі пасажири

повинні пересадитися з D на C. Літак E може сісти лише на внутрішню смугу. Жоден з двох літаків не можуть зарезервувати однаковий часовий інтервал для однієї злітно-посадкової смуги.

Комп’ютерний практикум №5. Пошук в умовах протидії

Хід виконання роботи

1. Згідно варіанту побудувати дерево пошуку.
2. Реалізувати алгоритм пошуку в дереві Монте-Карло, мінімаксний алгоритм та алгоритм альфа бета відтинання.
3. Застосувати алгоритми до вашої задачі. Визначити час роботи алгоритму та кількість розгорнутих вершин.
4. Порівняти алгоритми, результати звести в таблицю

Короткі теоретичні відомості

Пошук в умовах протидії це пошук в мультиагентному середовищі, де кожний агент прагне підвищити власний добробут.

Формулювання проблеми гри:

- Початковий стан включає позицію на дошці і визначає гравця, який повинен ходити.
- Модель переходу RESULT(s, a), визначає результат дії a виконаної в стані s.
- Перевірка термінального стану, яка визначає, що гра закінчена. Стани, в яких гра закінчена, називаються термінальними станами .

Функція корисності повідомляє числове значення термінальних станів.

Оптимальна стратегія призводить, щонайменше, до такого ж сприятливого стану, як і будь-яка інша стратегія, в тих умовах, коли доводиться грати з противником, що не допускає помилок.

Дерево гри (рис.1): вершини можливі стани, дуги - дії, які може виконати агент.

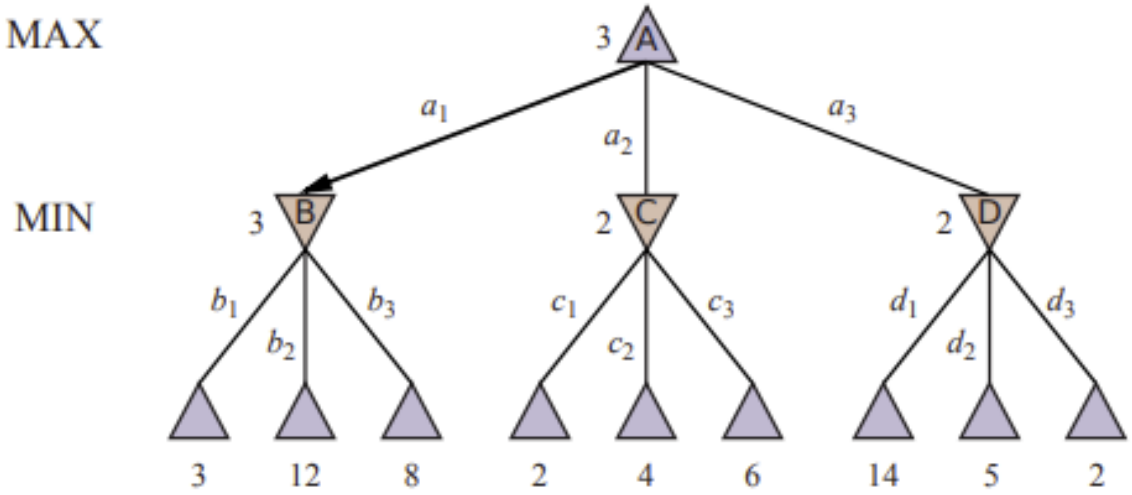


Рис. 1 Дерево гри

При наявності дерева гри оптимальну стратегію можна визначити, досліджуючи мінімаксне значення кожного вузла (рис. 2). Мінімаксним значенням вузла є корисність (для MAX) перебування у відповідному стані, за умови, що обидва гравці роблять ходи оптимальним чином від цього вузла і до вузла, що позначає кінець гри.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

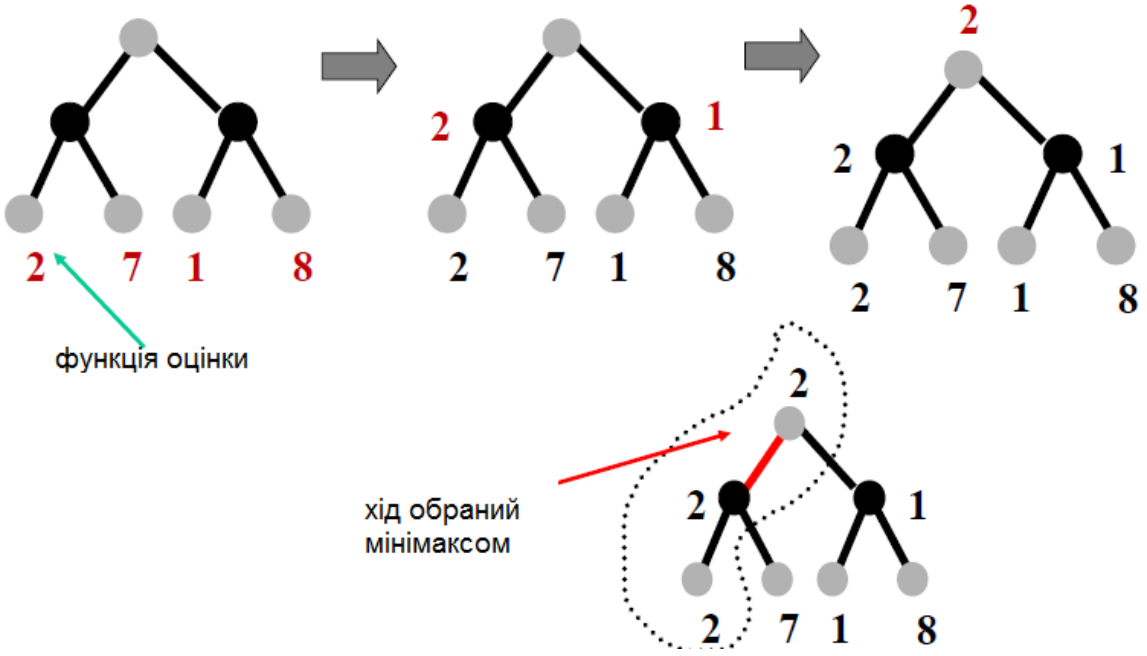


Рис. 2. Оптимальний хід обраний за стратегією мінімакса

Альфа бета відсікання така стратегія, що повертає такі ж ходи, які повернув би мінімаксний метод, але відсікає гілки, які по всій ймовірності, не здатні вплинути на остаточне рішення.

Алгоритм використовує значення альфа і бета.

- α = значення найкращого варіанту (тобто варіанта з найвищим значенням), який був досі знайдений в будь-якій точці вибору вздовж шляху для гравця MAX;
- β = значення найкращого варіанту (тобто варіанта з найнижчим значенням), який був досі знайдений в будь-якій точці вибору вздовж шляху для гравця MIN.

Алгоритм альфа-бета-пошуку в процесі своєї роботи оновлює значення α та β , а також відсікає гілки що залишилися в вузлі, як тільки стає відомо, що значення поточного вузла гірше в порівнянні з поточним значенням α або β для гравця MAX або MIN відповідно.

Приклад 1. Нехай в нас є наступне дерево гри Рис. 3. Виконати альфа бета відтинання.

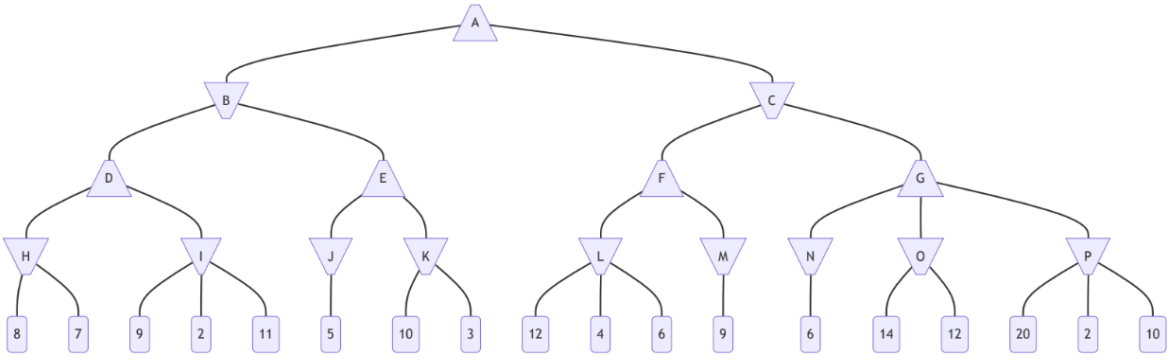


Рис.3

На початку [α, β]=[-∞, +∞]. Розгортаємо термінальні вузли по черзі, поступово оновлюючи значення α та β.

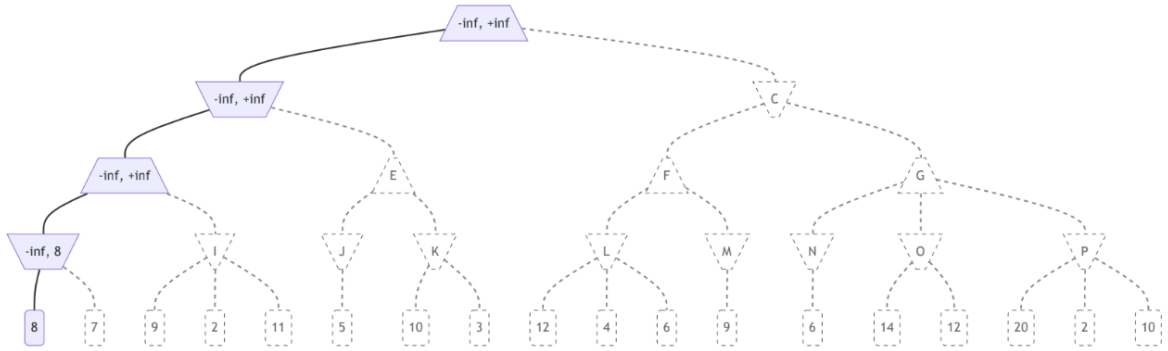


Рис.4.

На першому кроці спочатку β=8 (рис.4). Після розгортання наступного вузла β=7. І в вузлі H вже буде остаточне значення 7. Позначимо це як [7,7]. В батьківському вузлі D оновлюємо значення α=7 (рис.5).

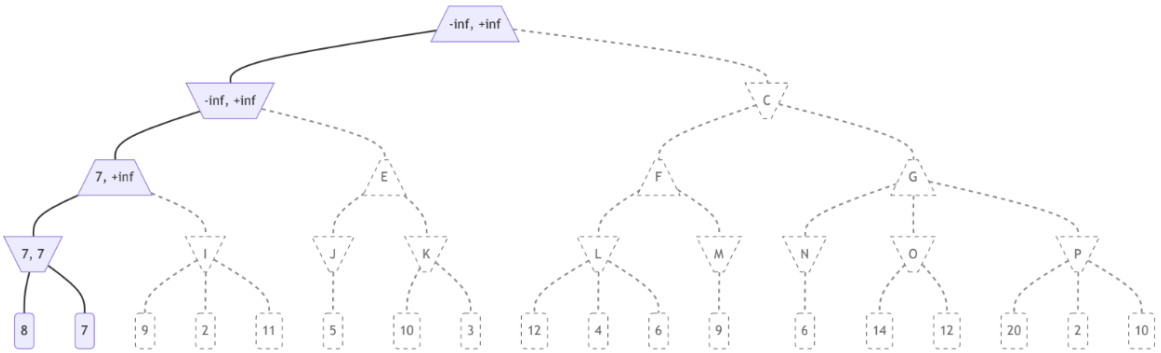


Рис.5

Знову розгортаємо термінальні вузли по черзі. Спочатку β=9. В батьківському вузлі є α і α< β (Рис. 6).

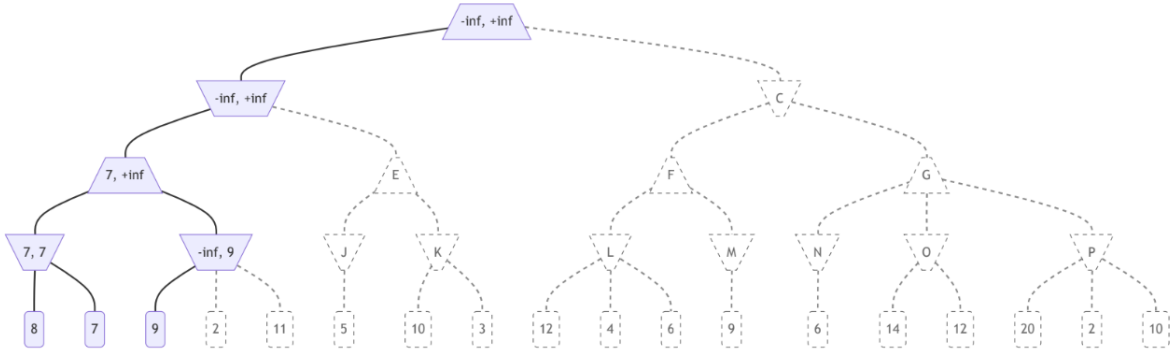


Рис.6

Розгортаємо наступний вузол β=2. В батьківському вузлі є α=7 і α>β. Тож

термінальний вузол зі значенням 11 не розгортається. В вузлі I буде значення менше або дорівнює 2, А для батьківського вузла В відомо що десь є варіант зі значення 7. На цьому кроці приймає рішення щодо дії гравець МАКС, і він обере дію, що веде до більшого значення 7. І розглядати всі подальші вузли після отримання значення 2 немає сенсу. В результаті в вузлі В маємо остаточне значення 7. Можна оновити значення β батьківського вузла, β=7 (рис.7).

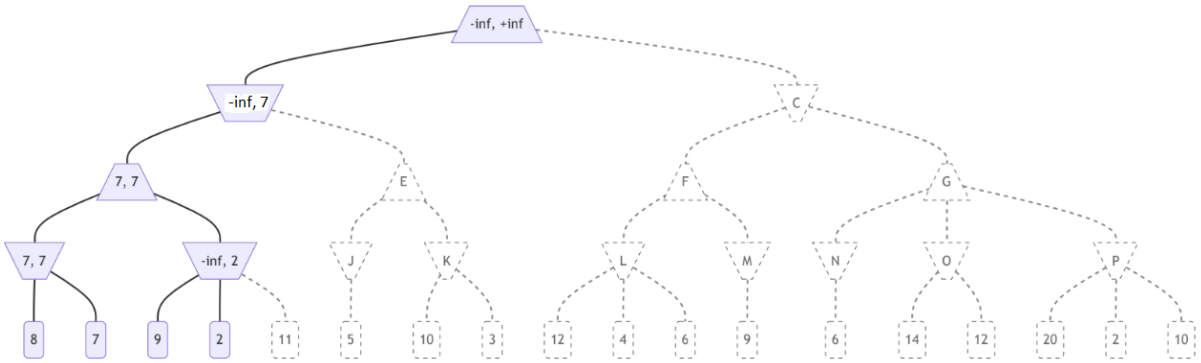


Рис. 7

Розгортаємо наступні термінальні вузли. В вузлі J є остаточне значення 5. Для вузла Е оновлюємо α=5. Порівнюємо зі значенням β=7 батьківського вузла α< β. Розгортаємо вузли далі (рис. 8).

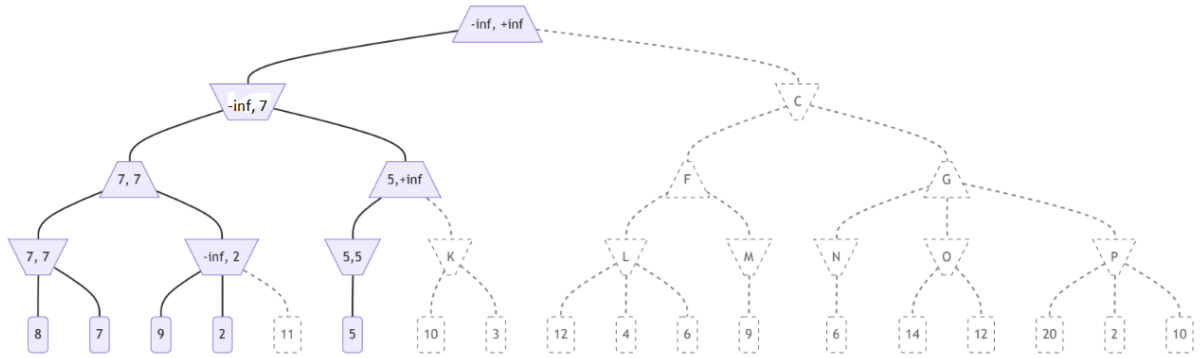


Рис.8

Розгортаємо термінальні вузли по одному. Оновлюємо значення β=10 (Рис.9). Порівнюємо α< β, розгортаємо далі.

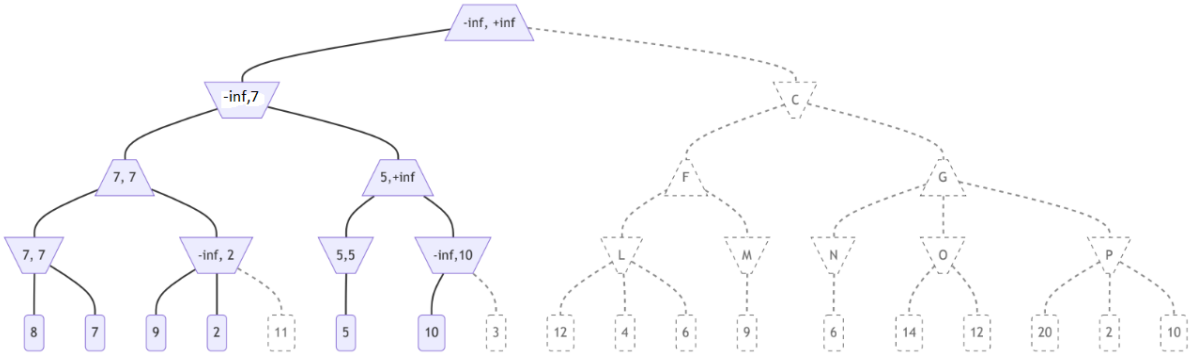


Рис.9

Отримаємо остаточні значення в вузлах К, Е, В. Оновлюємо α=5 в вузлі А (рис.10).

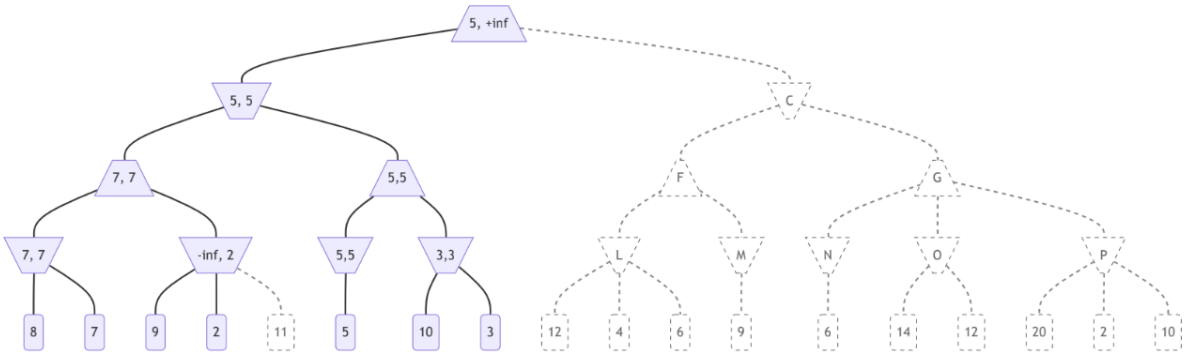


Рис.10

Розгортаємо наступну термінальну вершину і оновлюємо β=12. Перевіряємо найкращий варіант α та β, α< β (Рис.11).

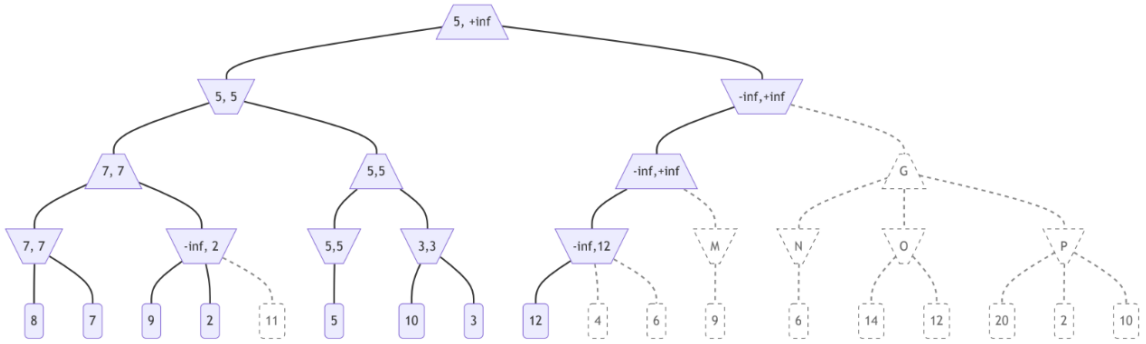


Рис.11

Розгортаємо наступний термінальний вузол і оновлюємо β=4. Перевіряємо найкращий варіант α=5 та β=4, α> β. Отже термінальну вершину зі значенням 6

не розгортаємо. В вузлі L буде значення не більше 4. Оновлюємо значення α=4 для вершини F (рис. 12).

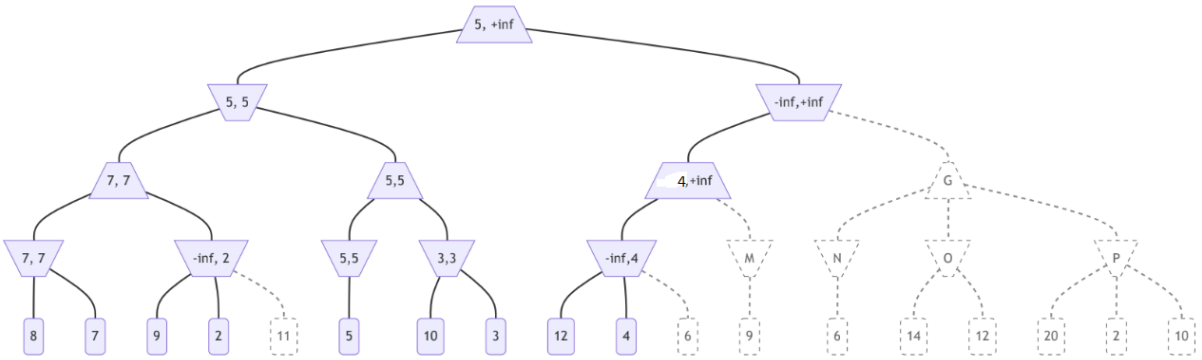


Рис.12

Розгортаємо вузли далі і отримаємо остаточне значення в вузлі F - це 9. Оновлюємо значення β=9 в вузлі C (рис.13).

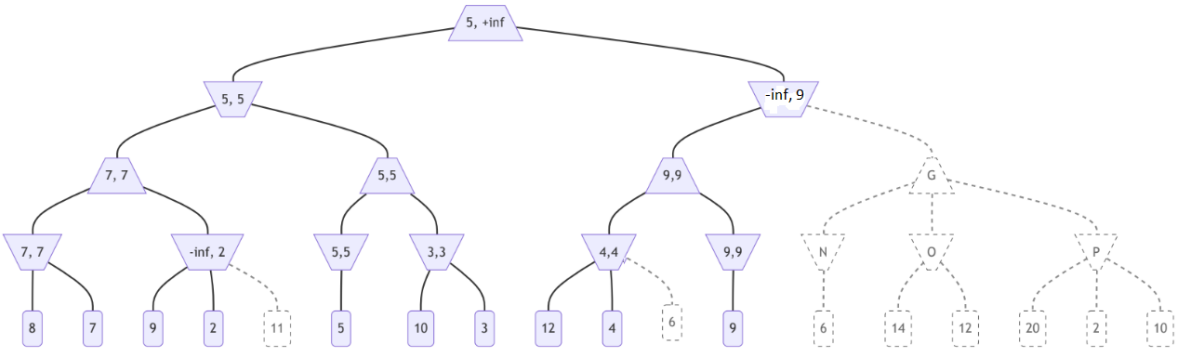


Рис.13

Розгортаємо вузли далі. Оновлюємо значення α=6 в вузлі G (Рис.14)

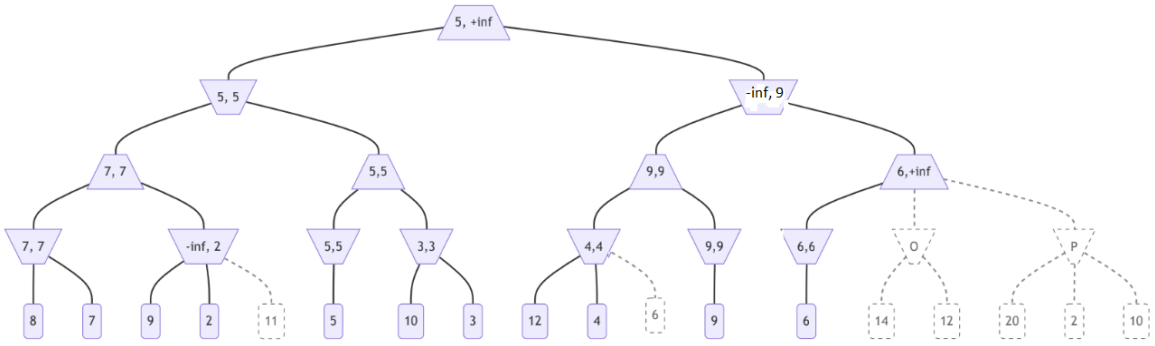


Рис.14

Розгортаємо вузли далі. Оновлюємо значення β=14 в вузлі O. Перевіряємо

що α> β (Рис.15).

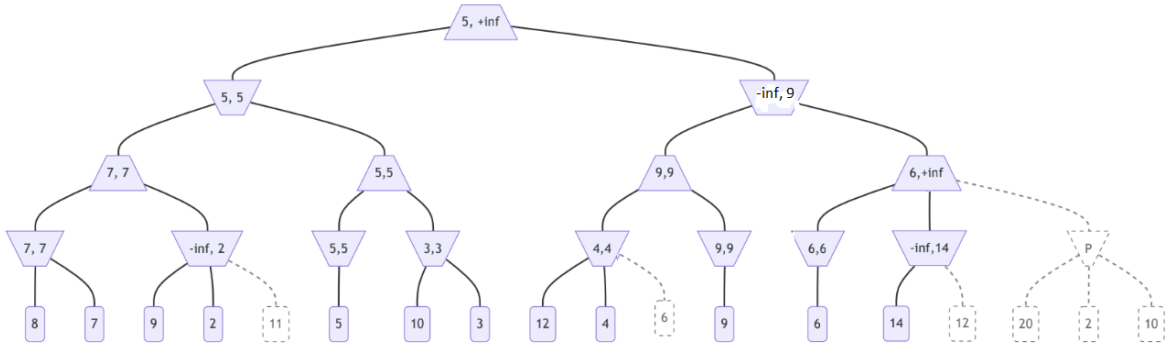


Рис.15

Розгортаємо термінальний вузол зі значенням 12. Маємо остаточне значення в вузлі O. І оновлюємо значення α=12 в вузлі G (Рис.16).

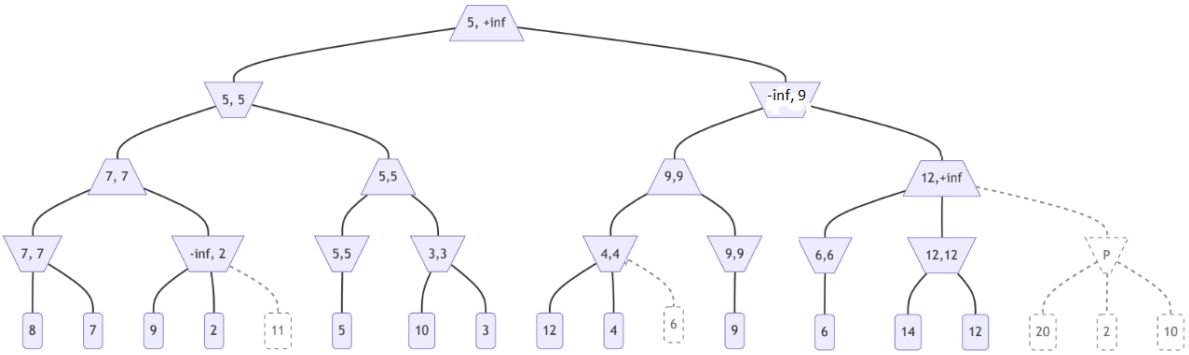


Рис.16

Порівнюємо α та β. Бачимо що α=12 > β=9. Отже вузол P і всі його дочірні вузли не розгортаються. В вузлі G буде значення не менше 12, отже в вузлі C остаточне значення 9. І в вузлі A значення 9 (Рис.17).

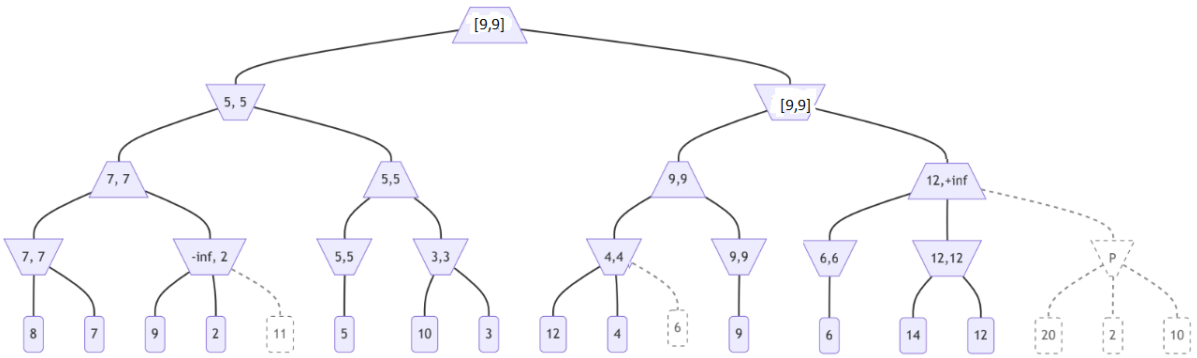


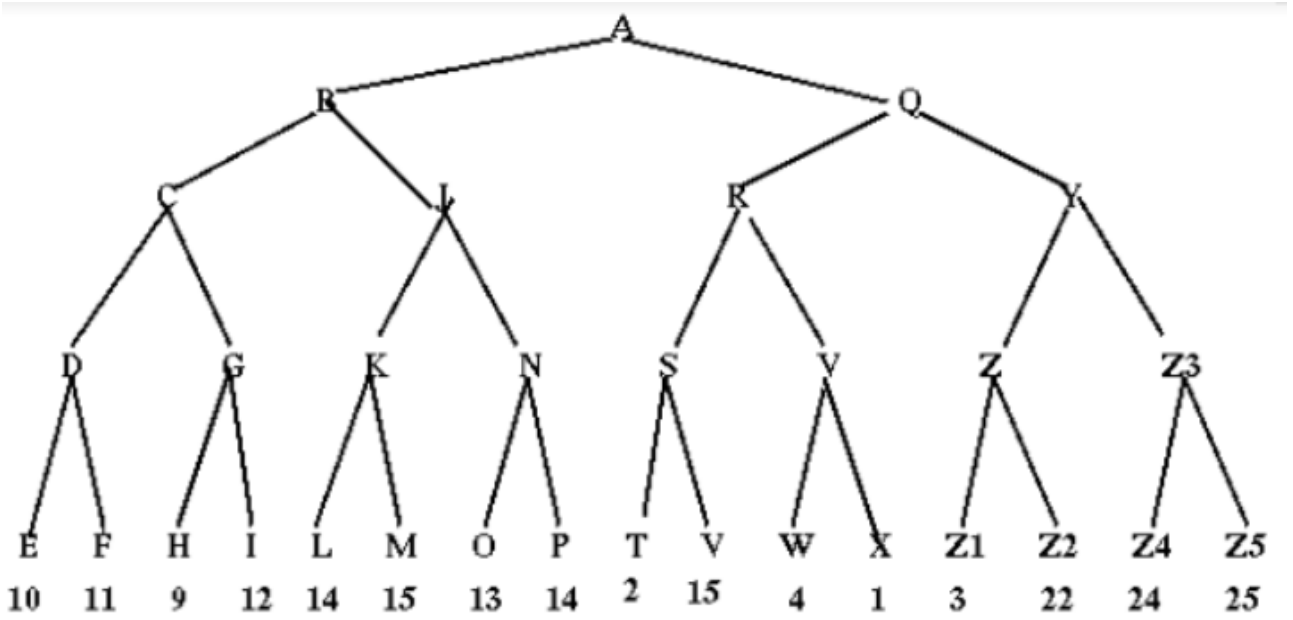
Рис. 17

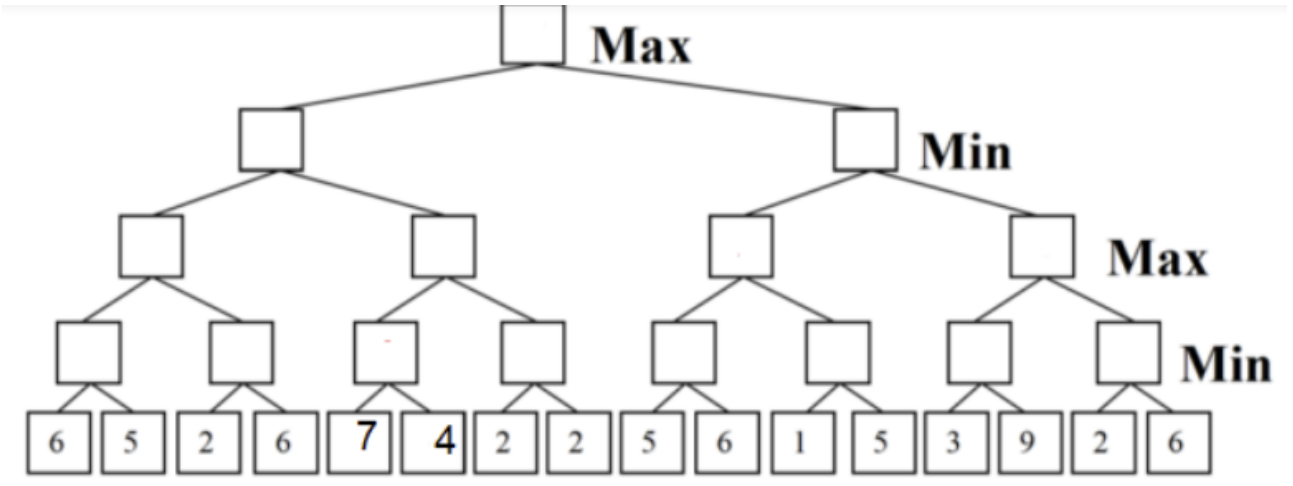
Контрольні запитання

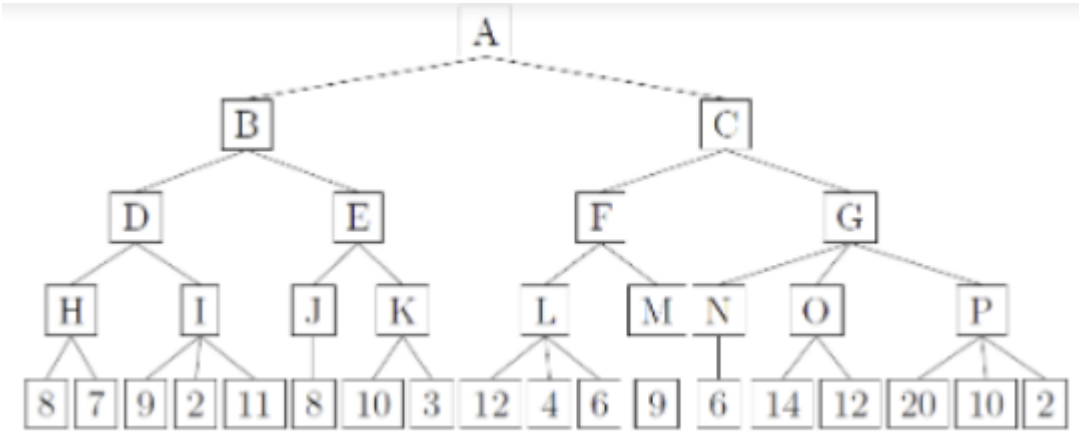
20. Як працює алгоритм мінімакс?
21. В чому перевага алгоритму альфа бета відтинання?
22. Навіщо потрібні параметри альфа та бета?
23. Які є підходи для того щоб не відвідувати термінальні вузли?
24. Які є етапи в пошуку в дереві Монте-Карло?

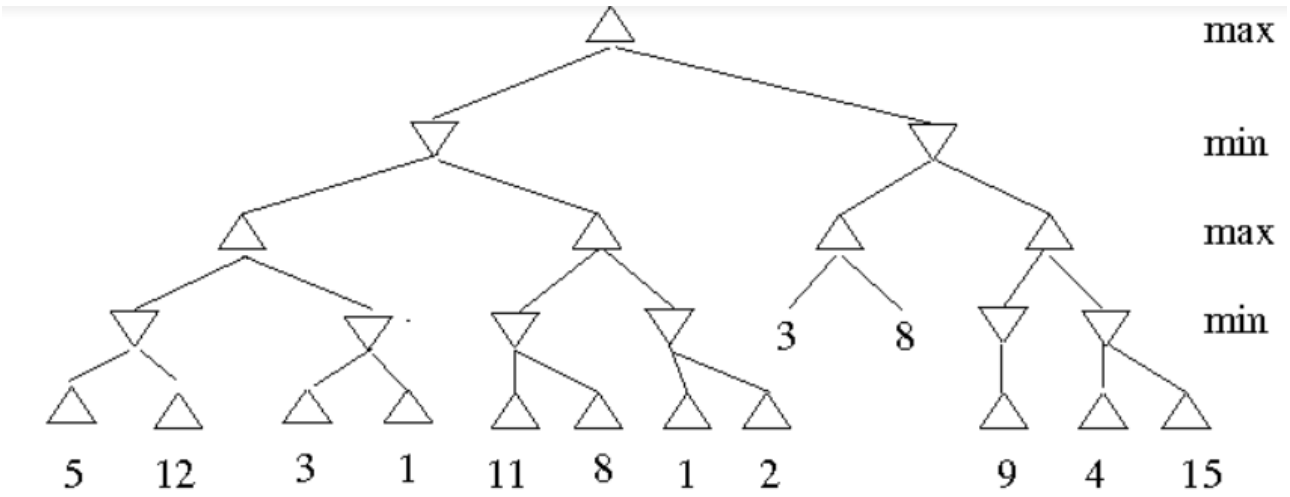
Задачі та вправи

Виконати алгоритм мінімакс та альфа бета відтинання:

1. 

2. 

3. 
A
B C
D E F G
H I J K L M N O P
8 7 9 2 11 8 10 3 12 4 6 9 6 14 12 20 10 2

4. 
max
min
max
min
3 8
5 12 3 1 11 8 1 2 9 4 15

Список використаної літератури

1. Stuart Russell, Peter Norvig. Artificial Intelligence: A Modern Approach 4rd Edition, - Upper Saddle River, NJ : Prentice Hall, 2021, 1166 p.