МОДЕЛІ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТА

Лекція 4

ІНФОРМОВАНИЙ ПОШУК ТА ДОСЛІДЖЕННЯ ПРОСТОРУ СТАНІВ

Зміст

- Жадібний алгоритм
- Пошук A*
- Рекурсивний пошук за першим найкращим збігом
- Пошук A* із обмеженням пам'яті
- Допустимі евристики

Пошук в дереві

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Стратегія визначається вибором порядку розкриття вузлів

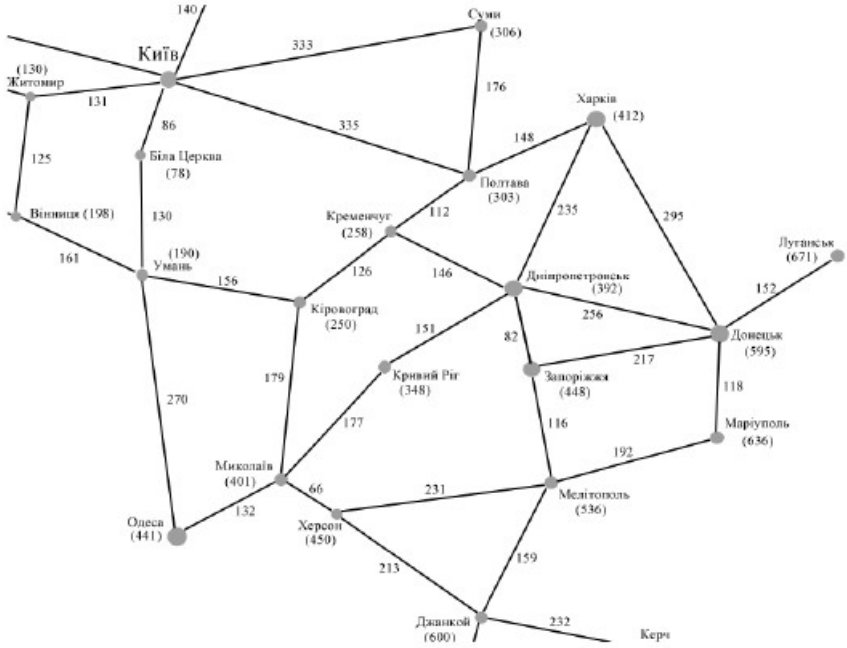
Інформативний пошук

- Загальний підхід: пошук за першим найкращим збігом (best first search)
- Для розгортання вузлів у алгоритмах Tree-Search та Graph-Search використовується функція оцінки *f(n)*
- Реалізація: периферія – це черга, відсортована за зменшенням значення функції оцінки вузлів
- Евристична функція *h(n)* як частина функції оцінки *f(n)*
- Евристична функція *h(n)* – приблизне значення відстані від поточного вузла *n* до найближчої цілі

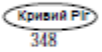
Жадібний алгоритм

- Пошук використовує тільки евристичну функцію: $f(n) = h(n)$
- Для подорожі Україною: відстань по прямій (Straight Line Distance – SLD) – h_{SLD}
- Жадібний алгоритм розкриває вузол, який видається знаходиться найближче до цілі

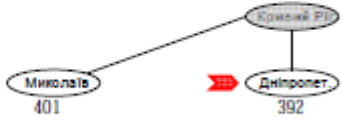
Місто	Відстань до Києва	Місто	Відстань до Києва
Біла Церква	78	Мелітополь	536
Вінниця	198	Миколаїв	401
Джанкой	600	Мукачево	607
Дніпропетровськ	392	Одеса	441
Донецьк	595	Полтава	303
Дубно	340	Рівне	304
Житомир	130	Севастополь	690
Запоріжжя	448	Сімферополь	666
Івано-Франківськ	452	Суми	306
Керч	719	Тернопіль	367
Кіровоград	250	Ужгород	628
Ковель	418	Умань	190
Кременчуг	258	Харків	412
Кривий Ріг	348	Херсон	450
Луганськ	671	Хмельницький	277
Луцьк	369	Чернівці	409
Львів	470	Чернігів	129
Маріуполь	636	Ялта	715


Київ
(130)
Житомир
Суми
(306)
Харків
(412)
Біла Церква
(78)
Полтава
(303)
Вінниця (198)
Кременчуг
(258)
(190)
Умань
Дніпропетровськ
(392)
Луганськ
(671)
Кіровоград
(250)
Донецьк
(595)
Кривий Ріг
(348)
Запоріжжя
(448)
Маріуполь
(636)
Миколаїв
(401)
Мелітополь
(536)
Одеса
(441)
Херсон
(450)
Джанкой
(600)
140
333
176
131
86
335
148
125
130
112
235
295
161
126
146
152
156
151
256
82
217
179
270
177
116
118
192
66
231
132
213
159
232

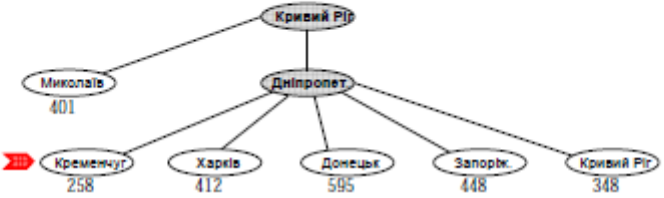
Приклад: жадібний алгоритм



Приклад: жадібний алгоритм



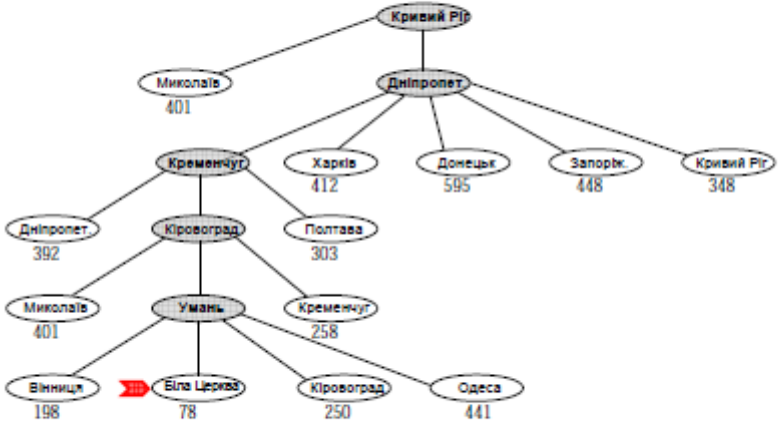
Приклад: жадібний алгоритм



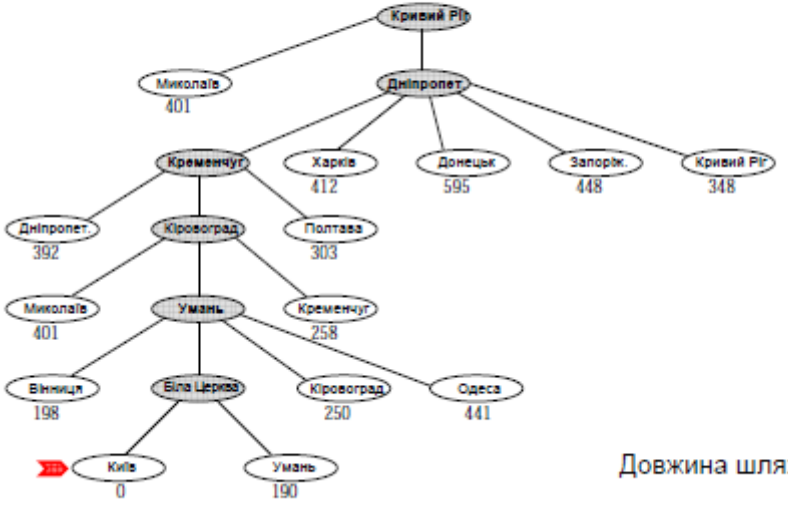
Приклад: жадібний алгоритм

Приклад: жадібний алгоритм

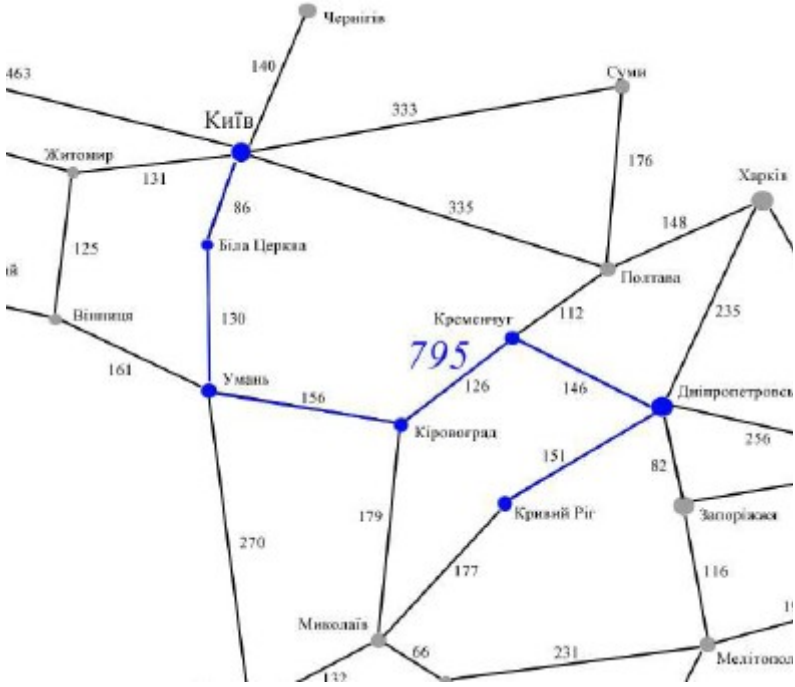
Приклад: жадібний алгоритм

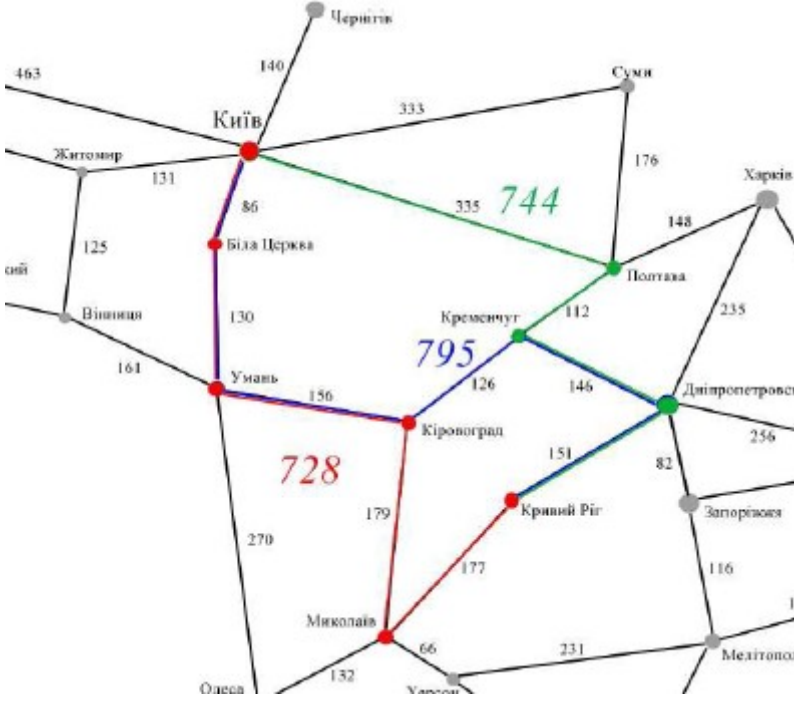


Приклад: жадібний алгоритм



Довжина шляху: 795


Чернігів
140
463
Суми
333
Київ
Житомир
131
176
Харків
86
335
148
125
Біла Церква
Полтава
Вінниця
130
Кременчуг
112
235
795
161
Умань
156
126
146
Дніпропетровськ
Кіровоград
256
151
82
179
Кривий Ріг
Запоріжжя
270
177
116
Миколаїв
66
231
Мелітополь


Чернігів
463
140
Суми
Київ
333
Житомир
131
176
Харків
86
335
744
148
125
Біла Церква
Полтава
Вінниця
130
Кременчуг
112
235
795
161
Умань
156
126
146
Дніпропетровс
Кіровоград
256
728
151
82
179
Кривий Ріг
Запоріжжя
270
177
116
Миколаїв
66
231
Мелітопол
132
Одеса
Херсон

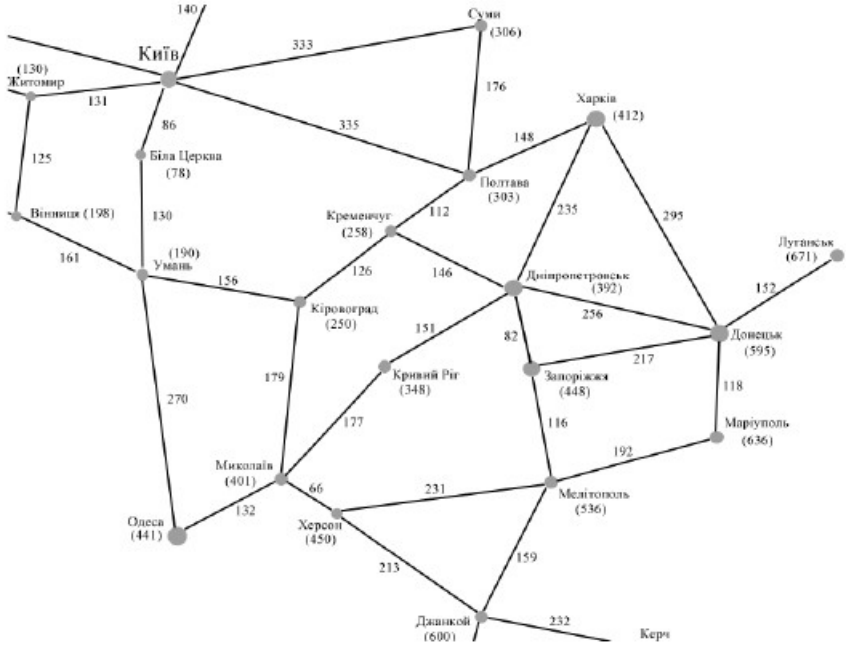
Властивості жадібного алгоритму

- Повний? Ні – може застрягати у циклах
- Час? $O(b^m)$, проте добра евристика значно покращує результат
- Простір? $O(b^m)$ - зберігає всі вузли в пам'яті
- Оптимальний? Ні

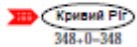
Алгоритм A*

- Ідея: запобігати розкриттю шляхів, які вже є дорогими
- Функція оцінки: $f(n) = g(n) + h(n)$
  - $g(n)$ – вартість досягнення поточного вузла n з початкового
  - $h(n)$ – оцінена вартість досягнення цільового вузла з n
  - $f(n)$ – оцінена повна вартість шляху через поточний вузол n
- A* є оптимальним, якщо $h(n)$ – допустима:
  - $h(n) \le h^*(n)$, де $h^*(n)$ – справжня вартість з n
    - тобто $h(n)$ ніколи не переоцінює відстань – є оптимістичною
  - $h(n) \ge 0$; $h(G) = 0$ для будь-якої цілі G
- Функція $h_{SLD}(n)$ ніколи не переоцінює дійсну відстань

- Теорема: пошук A* - оптимальний


Київ
(130)
Житомир
Суми
(306)
Харків
(412)
Біла Церква
(78)
Полтава
(303)
Вінниця (198)
Кременчуг
(258)
Умань
(190)
Луганськ
(671)
Дніпропетровськ
(392)
Кіровоград
(250)
Донецьк
(595)
Кривий Ріг
(348)
Запоріжжя
(448)
Маріуполь
(636)
Миколаїв
(401)
Мелітополь
(536)
Одеса
(441)
Херсон
(450)
Джанкой
(600)
140
333
176
131
86
335
148
125
130
112
235
295
161
126
146
152
156
151
82
256
217
179
118
270
177
116
192
66
231
132
213
159
232

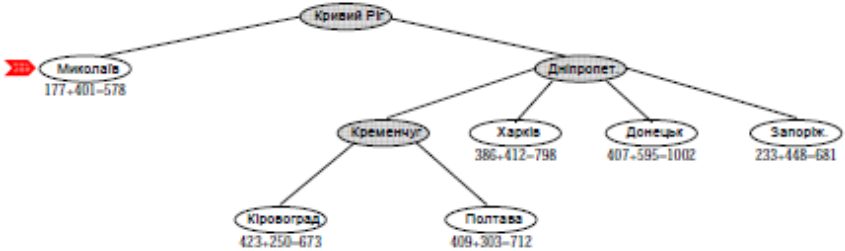
Приклад: алгоритм A*



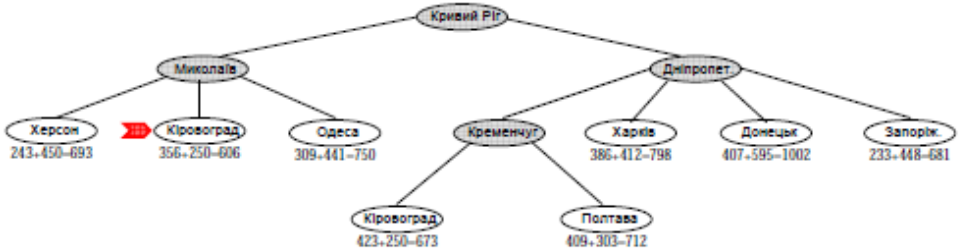
Приклад: алгоритм A*

Приклад: алгоритм A*

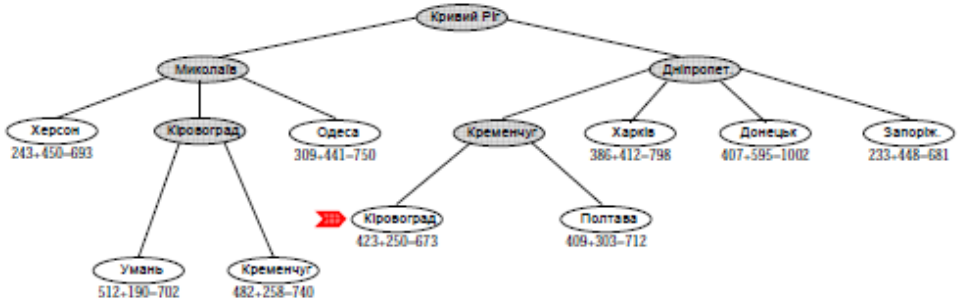
Приклад: алгоритм A*



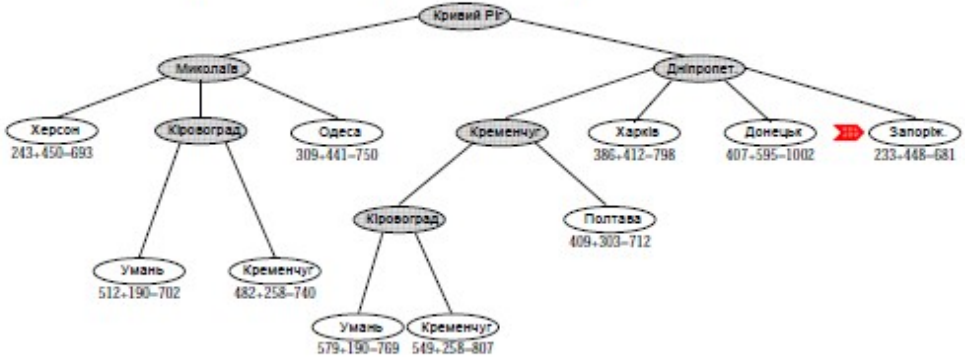
Приклад: алгоритм A*



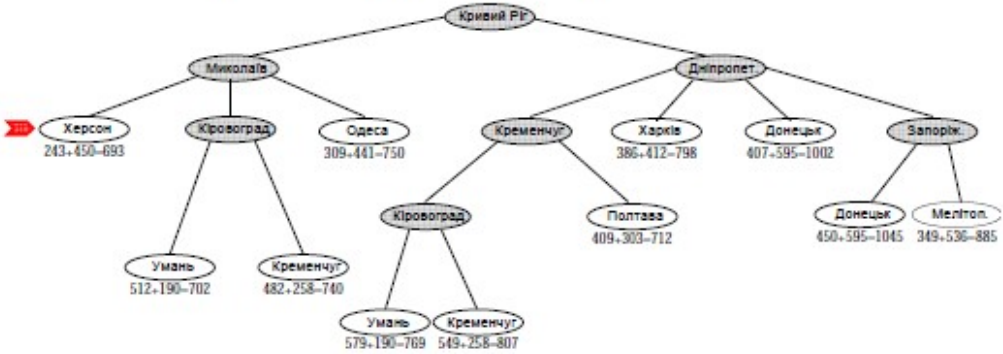
Приклад: алгоритм A*



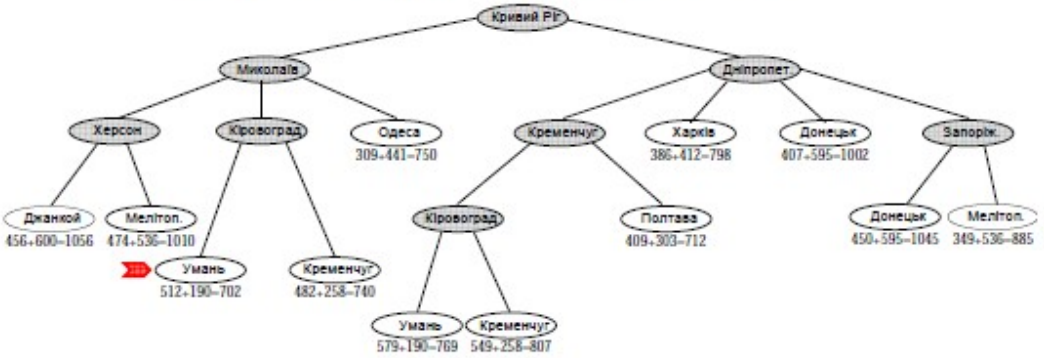
Приклад: алгоритм A*



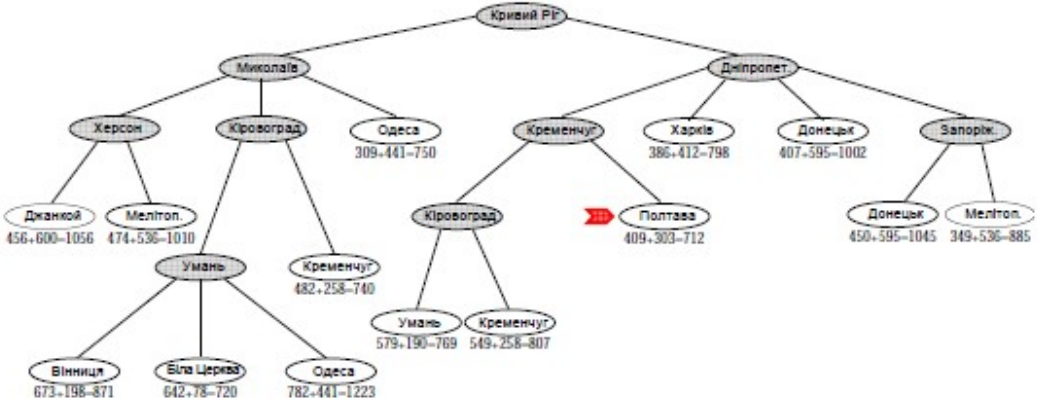
Приклад: алгоритм A*



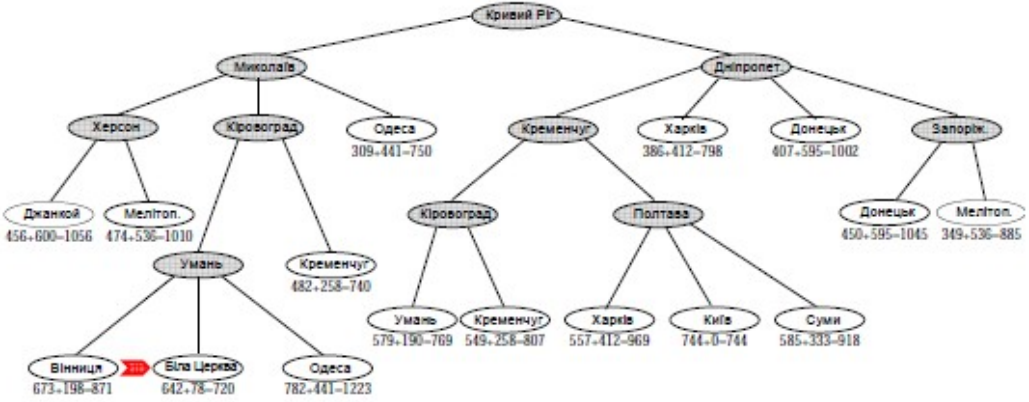
Приклад: алгоритм A*



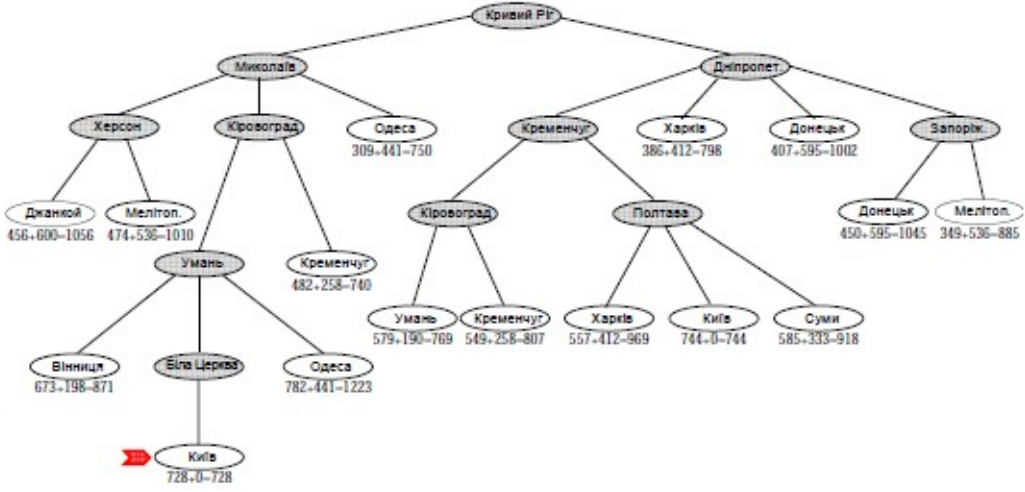
Приклад: алгоритм A*



Приклад: алгоритм A*

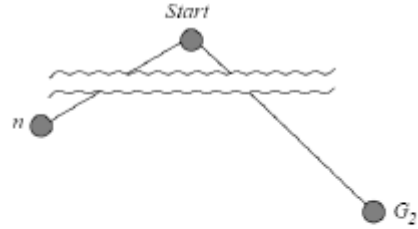


Приклад: алгоритм A*



Оптимальність A* (доведення)

- Припустимо, що є периферійний неоптимальний цільовий вузол G_2; $h(G_2)=0$. Вартість оптимального рішення = C^*.
- Нехай n – нерозвинений вузол на найкоротшому шляху до оптимальної цілі G_1



- $f(G_2) = g(G_2) + h(G_2) = g(G_2)$
- $f(G_2) > f(G_1)$: G_2 неоптимальний
- $f(n) = g(n) + h(n) \le C^*$
- $f(n) \le C^* \le f(G_2)$

  тобто вузол G_2 не буде розгортатись

A* для Graph-Search

- Graph-Search здатен відкинути оптимальний шлях до стану, який повторюється
  1) Алгоритм повинен відкидати найдорожчий з двох знайдених шляхів до одного й того самого вузла
  2) Перший шлях до будь-якого вузла завжди є оптимальним
     → функція $h(n)$ – спадкова (монотонна)

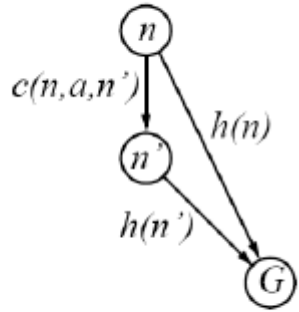
  Функція $h(n)$ є монотонною, якщо для будь-якого вузла n та його наступника n', який сформований за допомогою дії a, виконується:

  $$h(n) \leq c(n, a, n') + h(n')$$

- Теорема: пошук A* з використанням алгоритму Graph-Search є оптимальним, якщо $h(n)$ монотонна

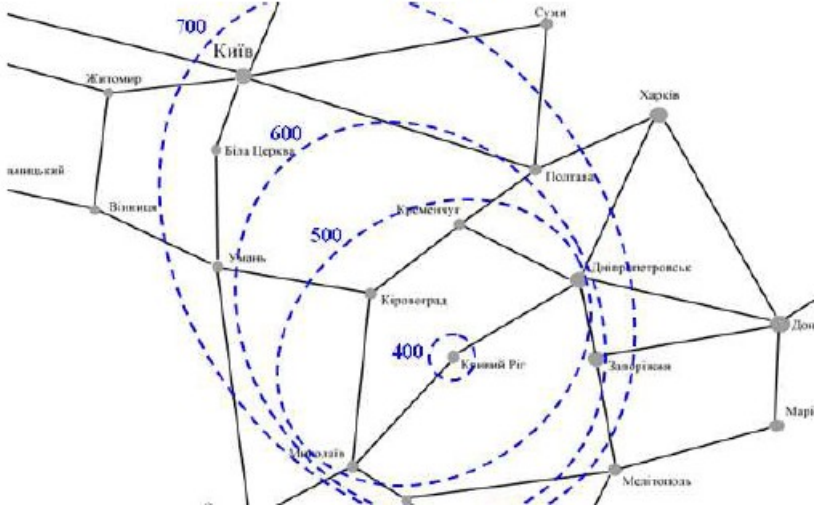
A* для Graph-Search: доведення

- $f(n') = g(n') + h(n') =$
  $= g(n) + c(n,a,n') + h(n') =$
  $\geq g(n) + h(n) = f(n)$
  $\rightarrow f(n') \geq f(n)$



- тобто функція f(n) не зменшується вздовж будь-якого шляху

F-контури для A*



Властивості A*

- Повний? Так
- Оптимальний? Так
  - Розгортає всі вузли з $f(n) < C^*$
  - Розгортає деякі вузли з $f(n) = C^*$
  - Не розгортає жодних вузлів з $f(n) > C^*$
  - Алгоритм A* є оптимально ефективним для будь-якої евристичної функції → для будь-якого іншого оптимального алгоритму не гарантується розгортання меншої кількості вузлів, ніж при пошуку A*!
  - Чи є пошук A* "срібною кулею"? - Ні.
- Час? Експоненційно залежить від довжини розв'язку
- Простір? Зберігає всі вузли в пам'яті

Властивості A*

- Під час роботи алгоритму A* відбувається відсікання гілок дерева, які є явно поганими (ведуть до неоптимальних розв'язків)
- Порівняйте алгоритм A* з:
  - Алгоритмом Дейкстри пошуку найкоротшого шляху в графі
  - Методом гілок та меж

Рекурсивний пошук за першим найкращим співпадінням

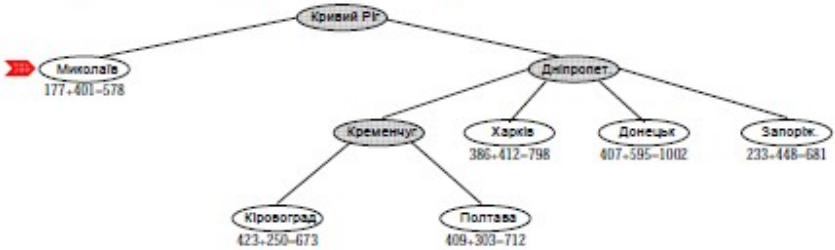
- Recursive Best-First Search – RBFS → стандартний пошук за першим-кращим, проте з використанням лінійного простору

Рекурсивний пошук за першим найкращим співпадінням

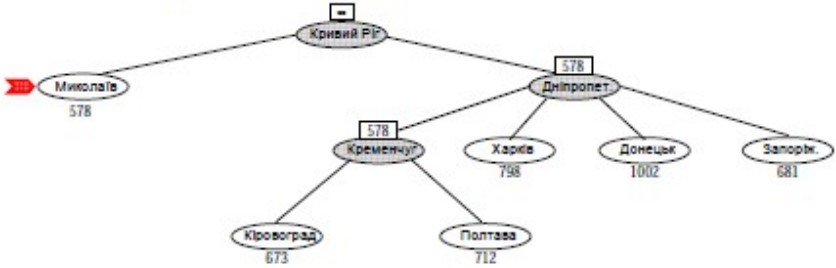
```
function Recursive-Best-First-Search(problem) returns рішення
result або індекатор невдачі failure
    RBFS(problem, Make-Node(Initial-State[problem]), ∞)

function RBFS(problem, node, f_limit) returns рішення result або
індикатор невдачі failure та нову межу f-вартості f_limit
    if Goal-Test[problem](State[node]) then return вузол node
    successors ← Expand(node, problem)
    if множина вузлів-наступників successors порожня
        then return failure, ∞
    for each s in successors do
        f[s] ← max(g(s)+h(s), f[node])
    repeat
        best ← вузол з найменшим f-значенням в множині successors
        if f[best] > f_limit then return failure, f[best]
        alternative ← друге після найменшого f-значення
                      в множині successors
        result, f[best] ← RBFS(problem, best,
                                  min(f_limit, alternative))
        if result ≠ failure then return result
```

Приклад: алгоритм A*

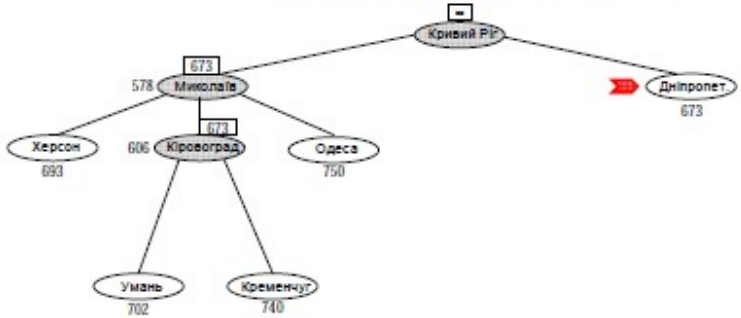


Приклад RBFS



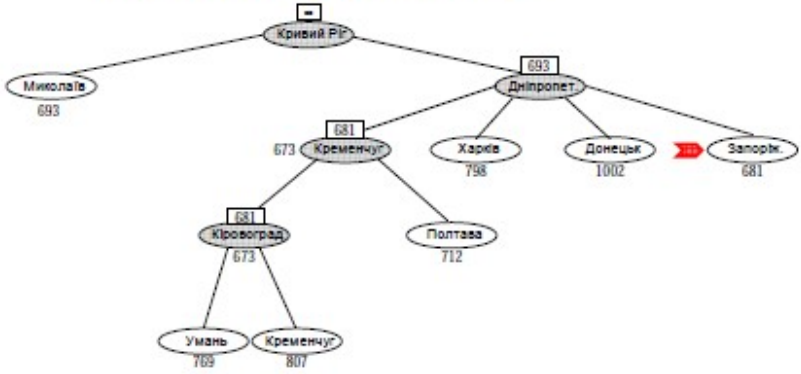
Ітерація №3
f_limit = 578

Приклад RBFS



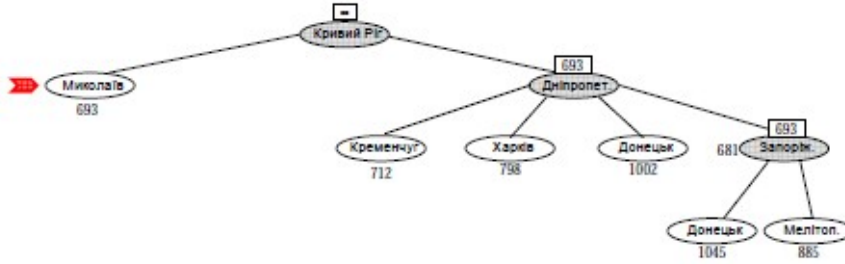
Ітерація №7
f_limit = 673

Приклад RBFS



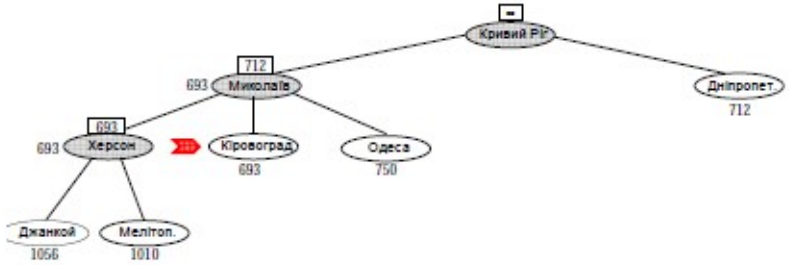
Ітерація №12
f_limit = 681

Приклад RBFS



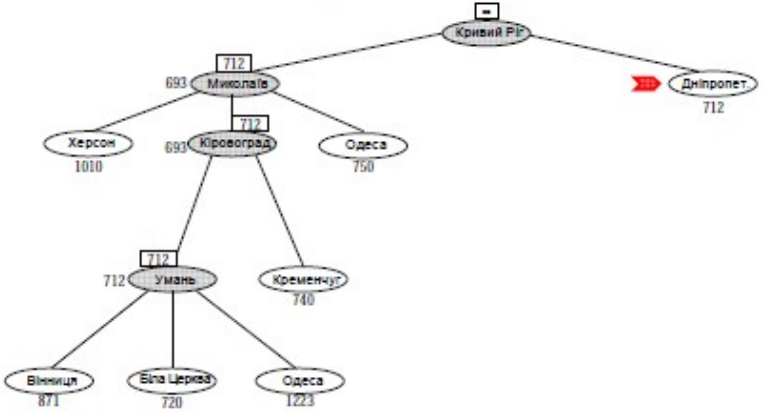
Ітерація №15
f_limit = 693

Приклад RBFS



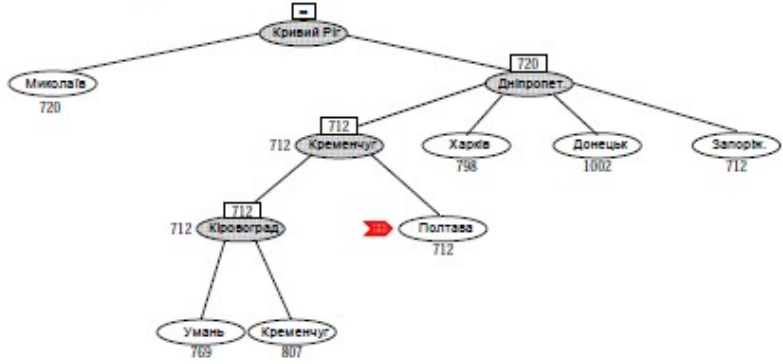
Ітерація №19
f_limit = 693

Приклад RBFS



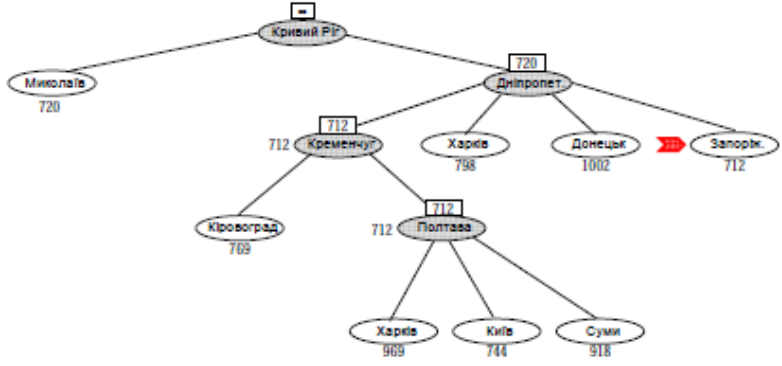
Ітерація №22
f_limit = 712

Приклад RBFS



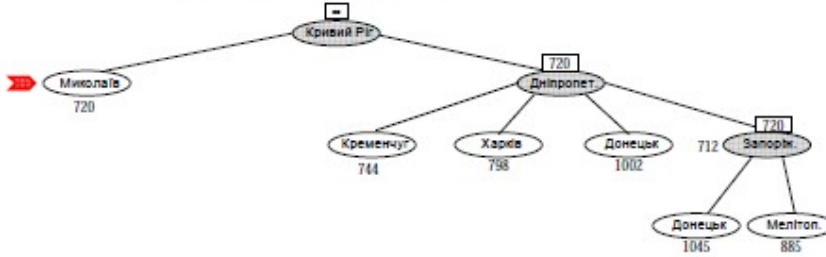
Ітерація №28
f_limit = 712

Приклад RBFS



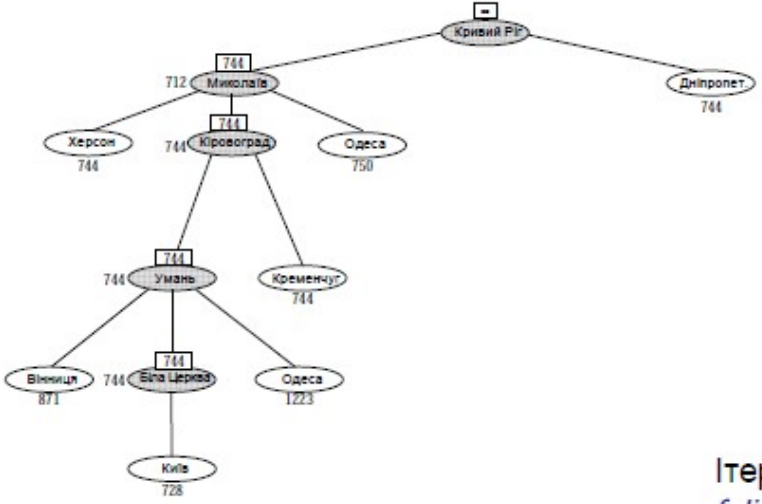
Ітерація №30
f_limit = 712

Приклад RBFS



Ітерація №33
f_limit = 720

Приклад RBFS



Ітерація №40
f_limit = 744

Властивості RBFS

- Повний? Так
- Час? Залежить від евристики та того, наскільки часто відбувались зміни найкращого шляху
- Простір? $O(bd)$
- Оптимальний? Так, якщо $h(n)$ допустима

- Алгоритм RBFS не може використати більше пам'яті ніж $O(bd)$ → Ідея: використовувати максимум наявної пам'яті → алгоритми MA* та SMA*

Пошук A* з обмеженням пам'яті

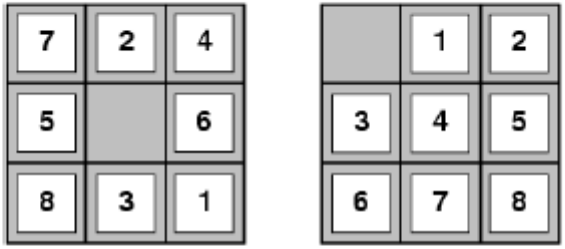
- SMA*: Simplified Memory-bounded A*
  - Працює аналогічно A*, розгортаючи найкращі вузли, допоки наявна вільна пам'ять
  - При відсутності вільної пам'яті видаляє найгірший листовий (периферійний) вузол та додається новий, кращий
  - При цьому значення видаленого вузла зберігається в батьківському вузлі (як в RBFS)
  - Якщо всі вузли в пам'яті однакові за оцінкою, то видаляється найстаріший з них
    - Видалений та доданий вузол можуть виявитись одним й тим самим, а отже вся пам'ять заповнена ланцюгом від кореня до поточного вузла → рішення знайти неможливо із заданим об'ємом пам'яті

Властивості SMA*

- Повний? Так, якщо d менша за об'єм пам'яті (виражений у вузлах)
- Оптимальний? Так, якщо це рішення можна досягнути

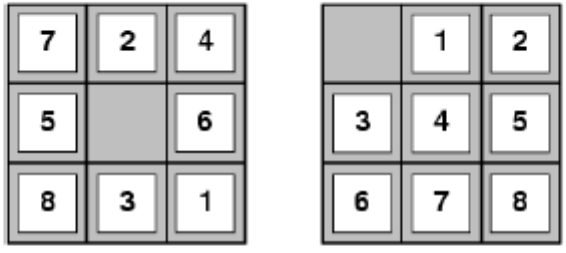
- Алгоритм SMA* може бути найкращим для пошуку оптимальних рішень, особливо якщо простір станів є графом, вартості етапів не однакові, а операції формування вузлів більш вартісні у порівнянні з утриманням списків вузлів
- Обмеження в об'ємі пам'яті можуть робити деякі задачі такими, що важко розв'язуються з точки зору часу обчислення

Допустимі евристики



- Середній коефіцієнт розгалуження ≈ 3
- В середньому задача розв'язується за 22 етапи
- Кількість вузлів при повному пошуку $3^{22} \approx 3{,}1 \cdot 10^{10}$
- Кількість різних станів = 9!/2 = 181 440

Допустимі евристики



- $h_1(n)$ – кількість фішок, які не стоять на своїх місцях
- $h_2(n)$ – сума відстаней всіх фішок до їх цільових станів (Манхетенська відстань)
- $h_1(S) = 8$
- $h_2(S) = 3+1+2+2+2+3+3+2 = 18$

Оцінка якості евристичних функцій

- Ефективний коефіцієнт розгалуження – b^*
  - Якщо загальна кількість вузлів, які генеруються під час пошуку A* для розв'язку конкретної задачі, дорівнює N, а глибина рішення – d, то b^* - коефіцієнт розгалуження, який повинно мати однорідне дерево з глибиною d та $N+1$ вузлом

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

  - Добре спроектовані евристичні функції мають b^* близький до 1

Порівняння значень вартості пошуку та ефективного коефіцієнта розгалуження для алгоритмів Iterative-Deeping-Search та A* з h_1, h_2

	Стоимость поиска			Эффективный коэффициент ветвления		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18094	1219	–	1,48	1,28
24	–	39135	1641	–	1,48	1,26

Домінування

- Якщо $h_2(n) \geq h_1(n)$ для всіх n (обидві є допустимими), тоді h_2 домінує над h_1 і її краще використовувати для пошуку
  - Кожний вузол, який повинен бути розгорнутим під час пошуку A* із h_2, буде також розгорнутим при застосуванні h_1; проте використання h_1 може призвести до розгортання й інших вузлів
- Для двох заданих допустимих евристик h_a та h_b

  $$h(n) = \max(h_a(n), h_b(n))$$

  також є допустимою та домінує над h_a та h_b

Послаблені задачі

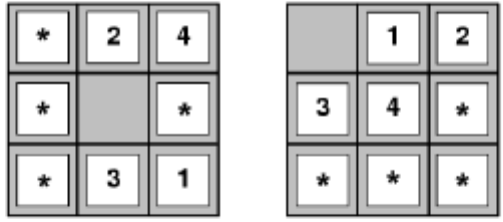
- Допустимі евристики можуть бути отримані як вартості точних рішень для послаблених задач
  - Якщо правила задачі гри в 8 послабити так, щоб фішка могла рухатись куди завгодно, а не тільки на сусідню вільну позицію, то h_1 повертала б точну кількість етапів у найкоротшому рішенні
  - Якщо будь-яку фішку можна було б рухати й на зайняті квадрати, то найкращий результат повертала б функція h_2
- Ідея: вартість оптимального рішення послабленої задачі є допустимою евристикою для початкової задачі
- Оптимальне рішення послабленої задачі не гірше за вартість оптимального рішення початкової задачі

Послаблені задачі

- Формальний запис правил задачі
  - *Фішка може бути переставлена з квадрату А в квадрат В, **якщо** квадрат А є суміжним по горизонталі або по вертикалі з квадратом В **та** квадрат В порожній*
- Варіанти послаблених задач
  - Фішка може бути переставлена з квадрату А в квадрат В, якщо квадрат А є суміжним до квадрату В
  - Фішка може бути переставлена з квадрата А в квадрат В, якщо квадрат В порожній
  - Фішка може бути переставлена з квадрату А в квадрат В

Допустимі евристики

- Можуть бути отримані на основі вартості рішення підзадачі конкретної задачі



- Бази даних з шаблонами містять точні вартості рішень для кожного можливого екземпляра підзадачі

Література