Лабораторна робота №1

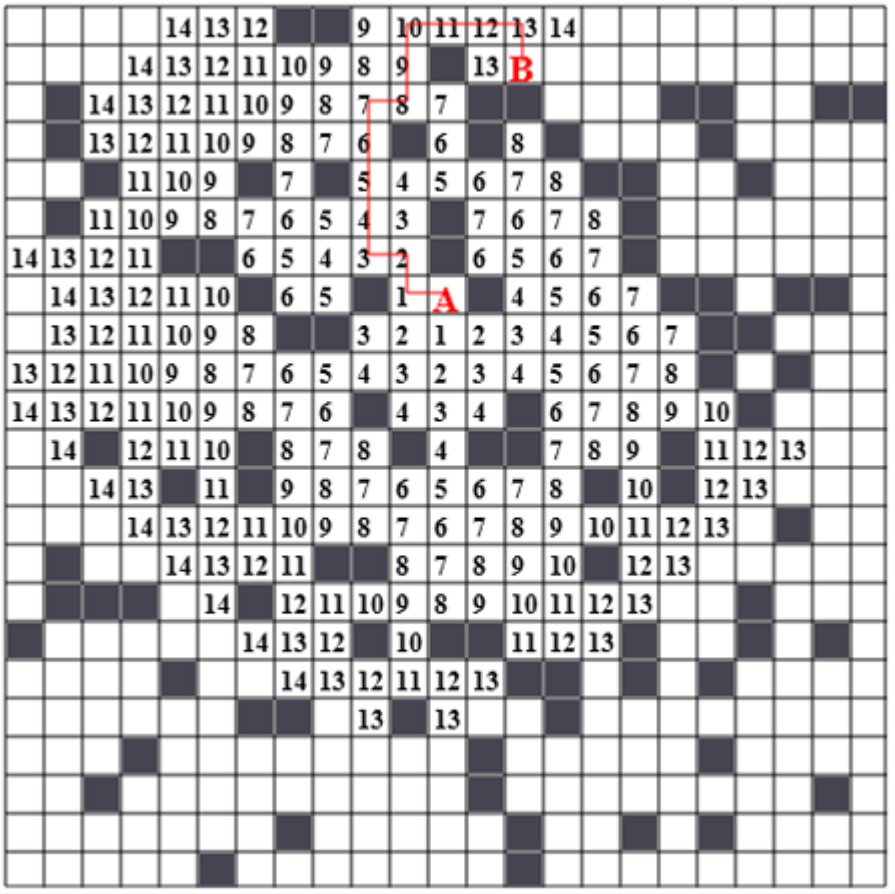
Алгоритми пошуку найкоротшого шляху на карті

Завдання: Написати код (мову програмування обираєте самі), для реалізації двох алгоритмів: алгоритму Лі та алгоритму пошуку A*, розібратися в роботі алгоритмів, вміти пояснити та аргументувати написаний код, відповдіати на запитання по теорії 1-2 лекцій.

Алгоритм Лі

Алгоритм Лі або хвильовий алгоритм – призначений для пошуку шляху на карті, дає змогу побудувати шлях або трасу між будь-якими елементами в лабіринті з довільною кількістю стін, дозволяє шукати найкоротший шлях на карті в комп'ютерних стратегічних іграх, належить до алгоритмів, що базуються на методах пошуку в ширину.

Поле розбивається на квадратні ділянки, розміри яких визначаються допустимою шириною трас. Процес пошуку поділяється на дві частини.



1. На двовимірній карті (матриці), яка складається з вільних клітинок, позначених 1, та заборонених (стін), позначених 0, позначається клітина старту (початковий елемент) та клітина фінішу. З початкового елемента відбувається поширення хвилі у чотирьох або восьми напрямках. Елемент, в який прийшла хвиля, утворює фронт хвилі. Кожна клітина, пройдена хвилею помічається, як пройдена. Кожен елемент першого фронту хвилі є джерелом вторинної хвилі. Хвиля в свою чергу не може проходити через клітинки, які помічені як

пройдені або як стіни. Елементи другого фронту хвилі генерують хвилю третього фронту і т.д. процес продовжується доти, доки не буде досягнуто кінцевий елемент або поки не стане зрозумілим, що його не можна досягти.

2. Будується траса. ЇЇ побудова відбувається від кінцевого елемента до початкового. Проте недоліком алгоритму є те, що при побудові траси необхідний великий об'єм пам'яті.

Існує два способи визначення кількості сусідніх клітинок: рахувати сусідні клітинки, доступні через їх спільні грані, а їх чотири або рахувати сусідні клітинки, доступні через спільні грані та спільні кути, а таких буде вісім.

Так, для першого варіанту, коли обрано чотири можливі напрямки руху:

```
  i+1
i+1  i  i+1
  i+1
```

Заповнення по матриці у восьми можливих напрямках, при цьому шлях по діагоналі приблизно в 1,4 рази більший:

```
i+3 i+2 i+3
i+2  i  i+2
i+3 i+2 i+3
```

При побудові траси рух відбувається залежно від обраних пріоритетів. Координати зменшуються при переході від початкового елемента до кінцевого. Ці пріоритети обираються в процесі розробки. Залежно від вибору тих чи інших пріоритетів отримуються різні траси, але довжина траси при цьому залишається однією і тією ж.

Робота цього алгоритму починається з початкового елемента. Біля нього розглядаються сусідні клітинки з чотирма або вісьмома можливими елементами. Від кожного з них до кінцевого елемента оцінюється довжина шляху, яку можна обчислити за формулою $C=|Ai-AD|+|Bi-BD|$, де (Ai, Bi) – координати однієї з сусідніх точок, (AD, BD) – координати останнього елемента. Із усіх знайдених значень довжини обирається найкоротше, тому елементом траси стає елемент, відстань від якого до кінцевої точки – найменша. Навколо цього елемента знову розглядаються сусідні чотири чи вісім елементів. Процес закінчується, коли буде досягнуто кінцевий елемент. Якщо на шляху зустрілися стіни, тобто рух в такий елемент заборонено, задається напрямок обходу стіни.

https://www.codesdope.com/blog/article/lee-algorithm/

Алгоритм A*

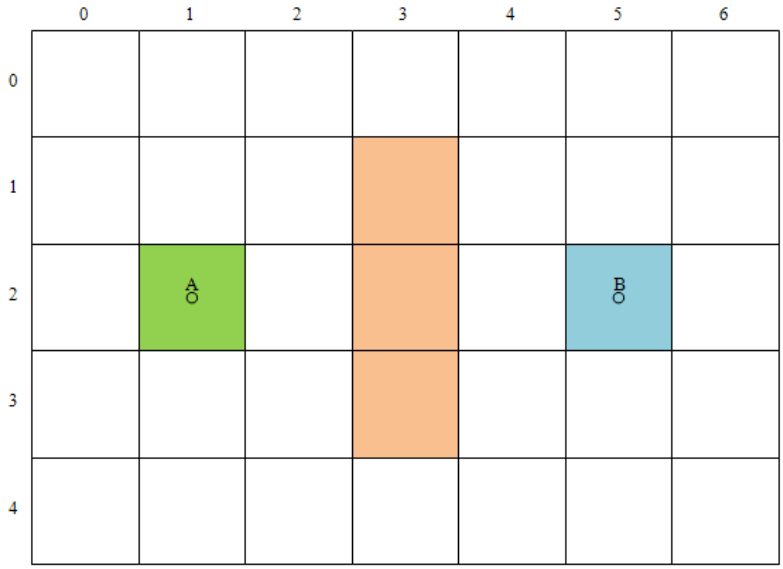
Алгоритм пошуку A* – призначений для пошуку найкоротшого маршруту, володіє двома ключовими характеристиками: оптимальністю, тобто гарантує отримання найкращого з можливий рішень, та повнотою, що означає, що алгоритм завжди знаходить рішення, якщо воно існує.

Порядок обходу вершин визначається евристичною функцією

$$f(x) = g(x)+h(x),$$

де g(x) – функція вартості шляху від початкової вершини до поточної, що розглядається на даному етапі,

h(x) – евристична функція оцінки, наближеної вартості шляху від поточної вершини x до кінцевої, ця функція не повинна переоцінювати відстань до кінцевої вершини.

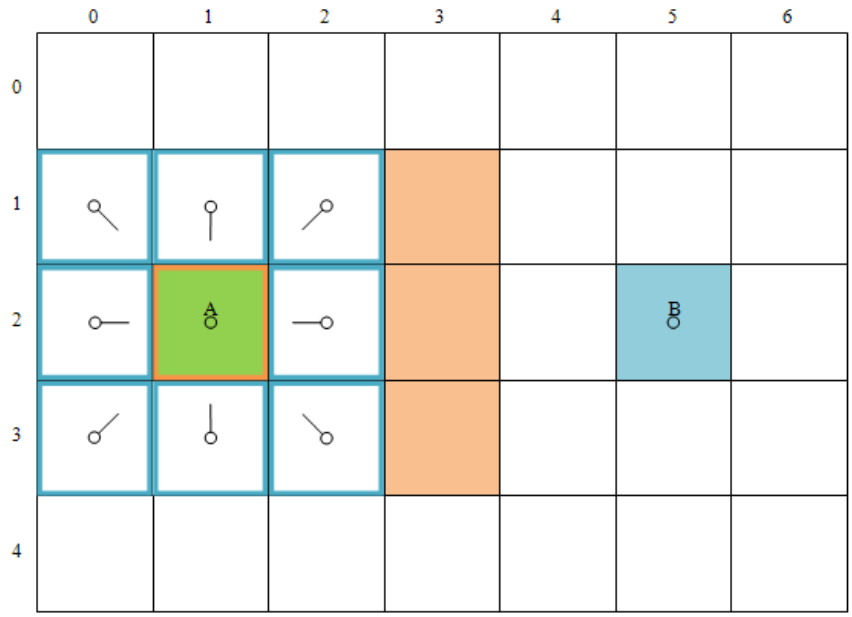


Нехай необхідно знайти найкоротшу відстань від початкової точки A до кінцевої – B. Область пошуку шляху розділена на квадратні комірки, що дозволить виразити область пошуку через двовимірний масив, кожен елемент якого буде представляти одну з клітинок. Кожна комірка при цьому помічається, як відкрита або заборонена (закрита).

1. Додати початкову клітинку A у список відкритих. В даний момент у цьому списку буде знаходитися лише одна комірка. Всі клітинки, що будуть потрапляти до відкритого списку – це клітинки, які потрібно перевірити та вирішити, чи будуть вони частиною шляху до кінцевої вершини.

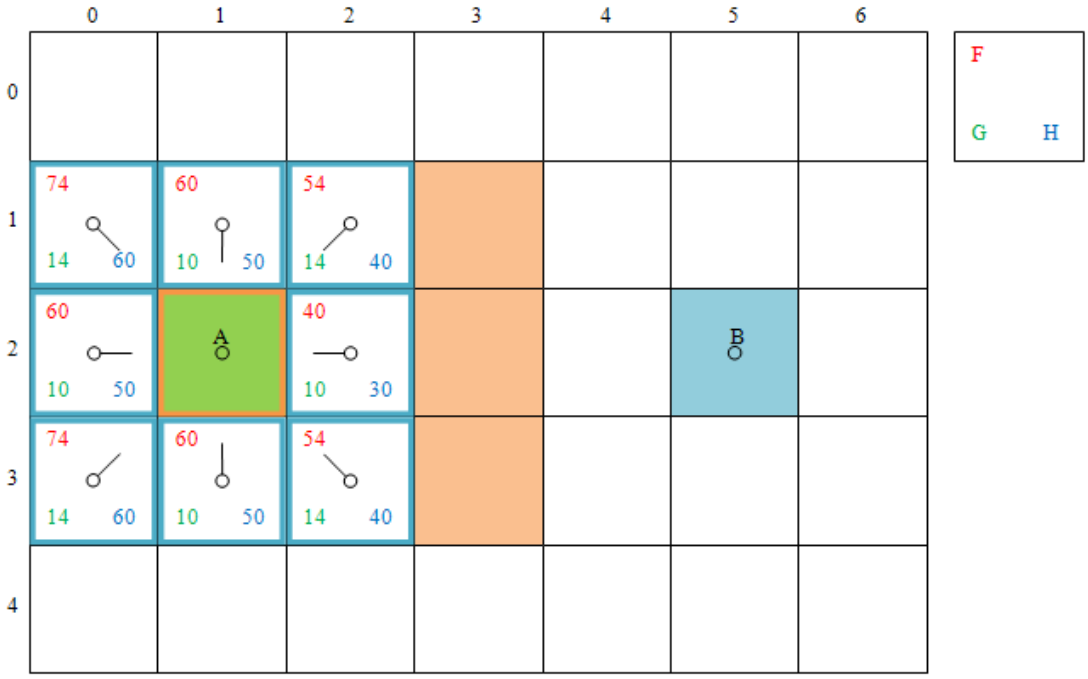
2. Знайти можливі вільні клітинки, що межують з початковою клітинкою, ігноруючи непрохідні (заборонені) клітинки, і додати можливі вільні у відкритий список. Для кожної з таких клітинок точка A являється батьківською.

3. Видалити початкову точку A з відкритого списку і додати у закритий список клітин, які більше не потрібно перевіряти.

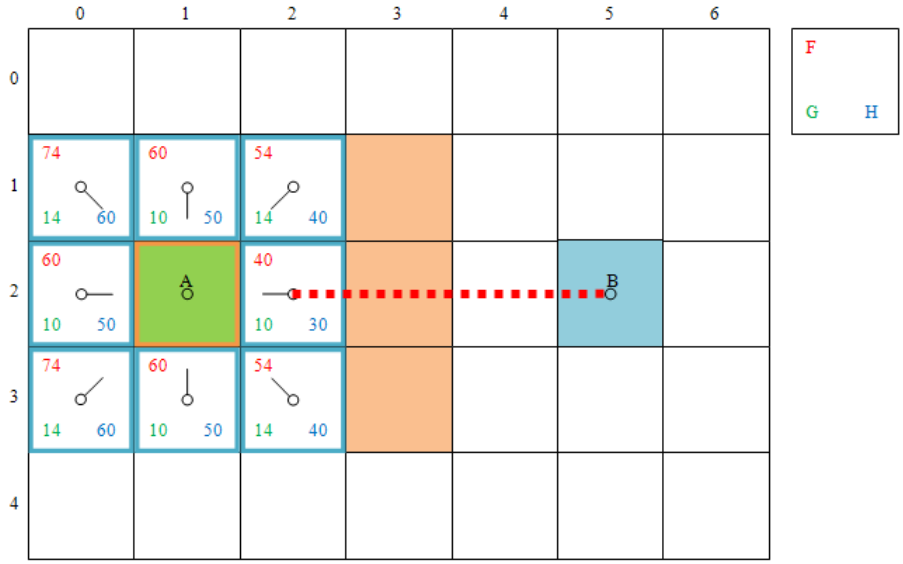


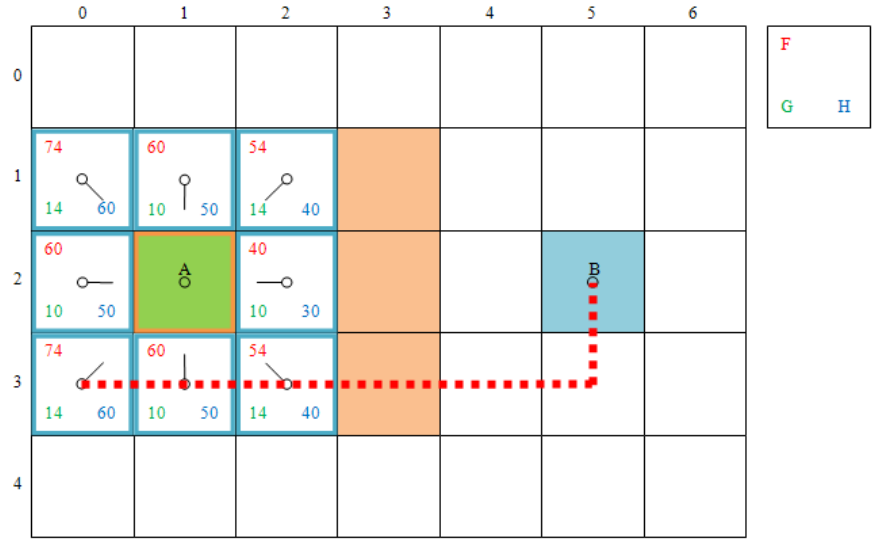
4. Обрати одну клітину, що знаходиться у відкритому списку і повторити з нею вище описаний процес. Клітинка обирається залежно від величини функції f(x).

Зазвичай відстань для проходження в сусідню клітину по горизонталі береться рівною 10, а по діагоналі - 14 одиниць. Для обчислення функції g(x) необхідно до величини g(x) її батьківської клітинки, тобто на даному кроці g(A) додати 10 або 14. Величина h(x) зазвичай обчислюється методом Манхеттена. Його зміст полягає у тому, щоб порахувати загальну кількість клітинок, необхідних для досягнення кінцевої клітинки B, від поточної клітини, що розглядається, ігноруючи діагональні переміщення між клітинками, а також будь-які заборони (стіни). Потім отримана кількість множиться на 10. Розрахувавши величну f(x) для всіх клітинок у відкритому списку, отримаємо:

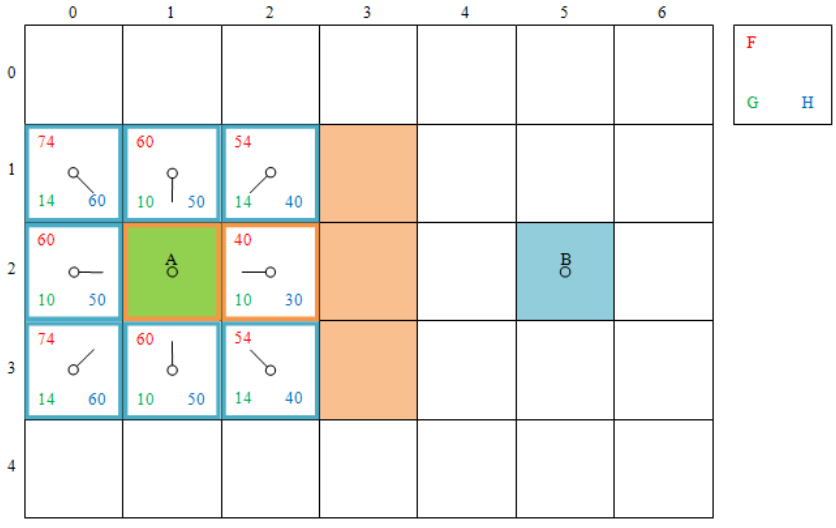


Розрахунок величини h(x) для клітинок з різними координатами:





5. Обирається комірка з відкритого списку з найменшим значенням функції f(x), це комірка з координатами [2,2], для якої f(x)=30.

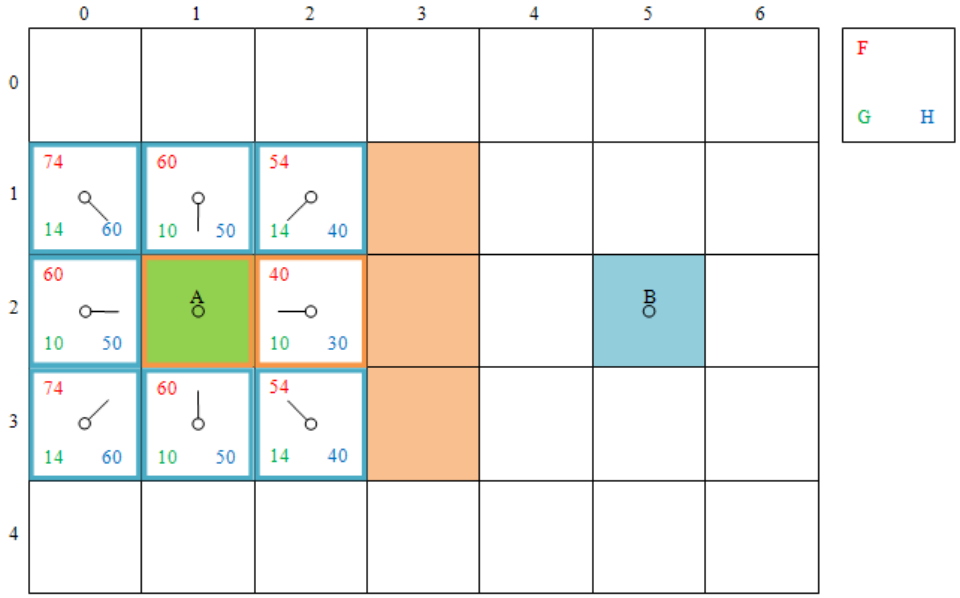


6. Обрана з відкритого списку клітинка [2,2] видаляється з нього і додається у закритий список.

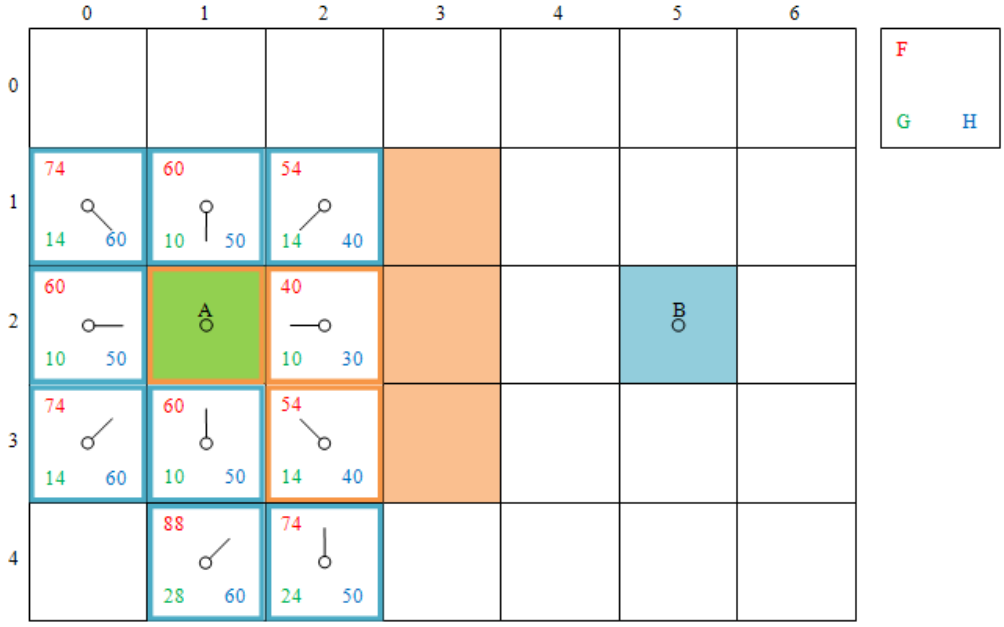
7. Додаємо у відкритий список всі сусідні з нею клітини, крім клітин закритого списку та заборонених (стін), вказуючи, що поточна вершина x є батьківською для них.

8. Обчислити значення g(x), h(x) та f(x) для кожної нової клітинки зі списку відкритих.

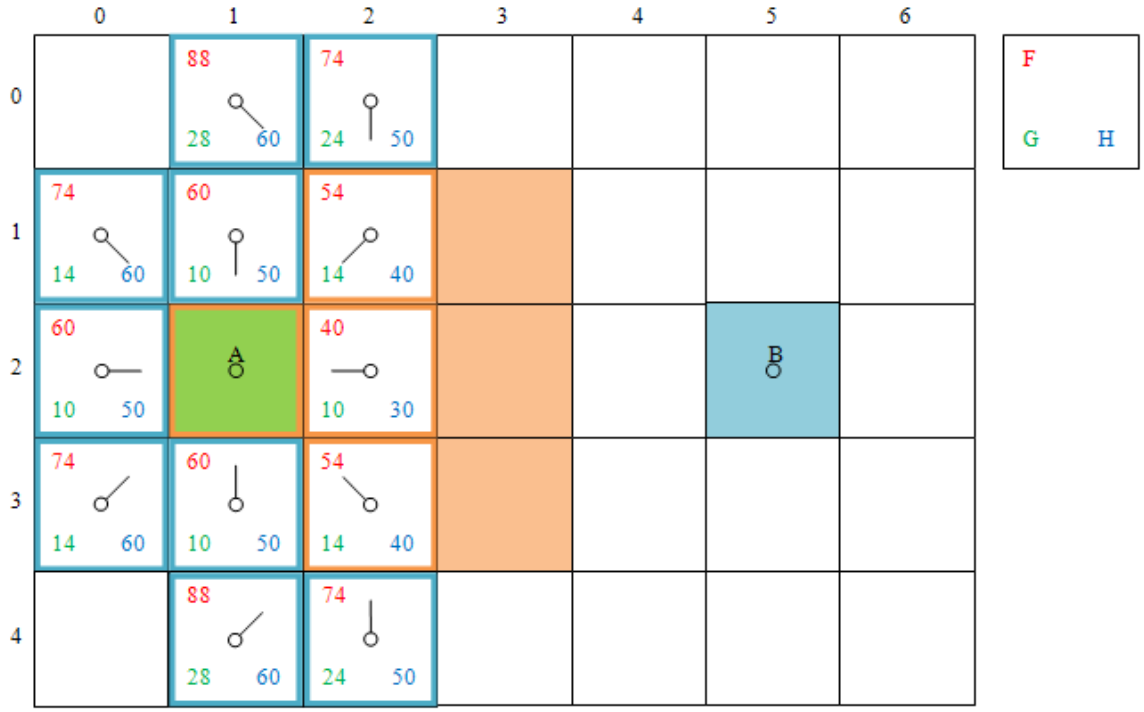
9. Якщо сусідня клітинка вже знаходиться у відкритому списку, то порівнюємо значення величини g(x) та поточної клітини, що перевіряється. Якщо попереднє значення менше нового, то нічого не відбувається, інакше в клітині у відкритому списку змінюється значення g(x) на нове, також змінюємо вказівник на батьківську клітинку, щоб він вказував на поточну клітинку.



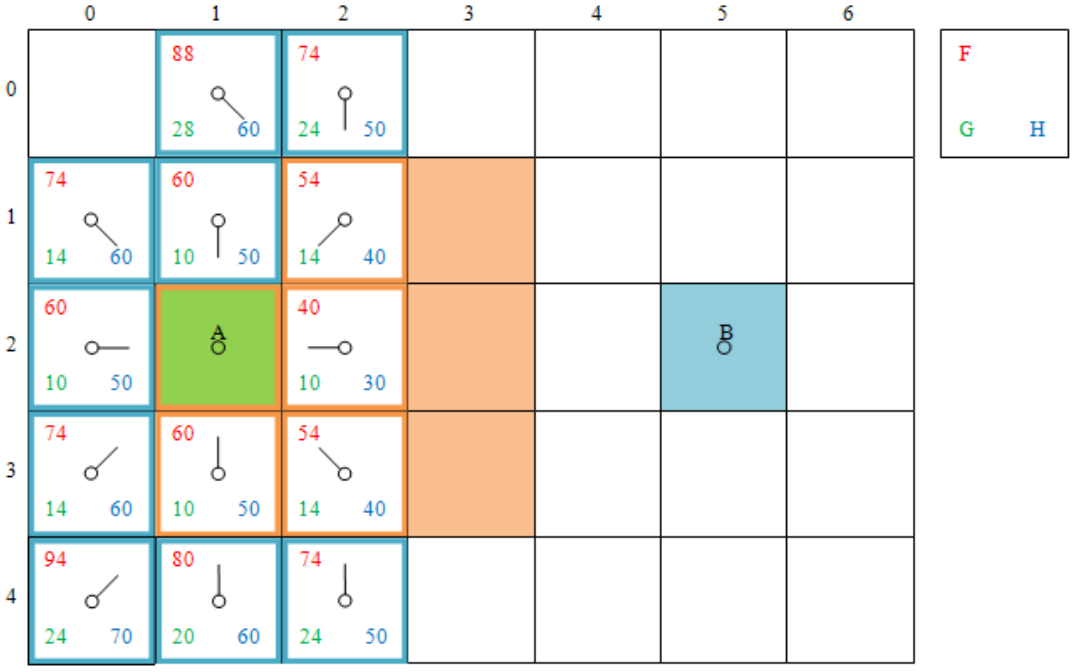
10. Тепер у відкритому списку знаходиться 7 клітин, дві з яких мають однакове значення функції f(x)=54, можна обрати довільну з них, обираємо нижню.



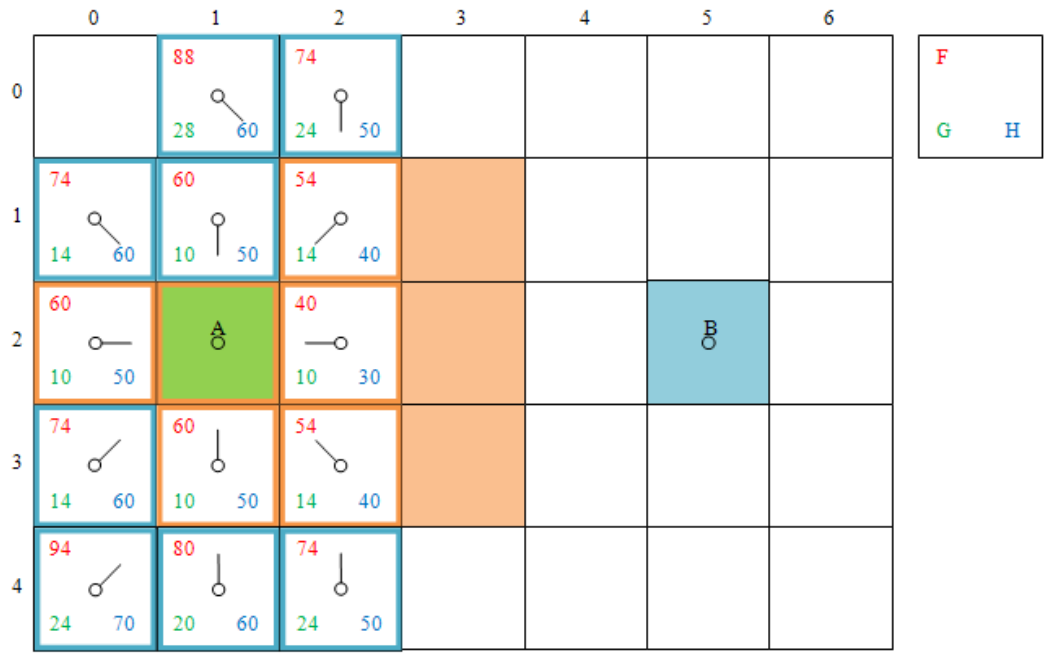
11. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [1,2]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



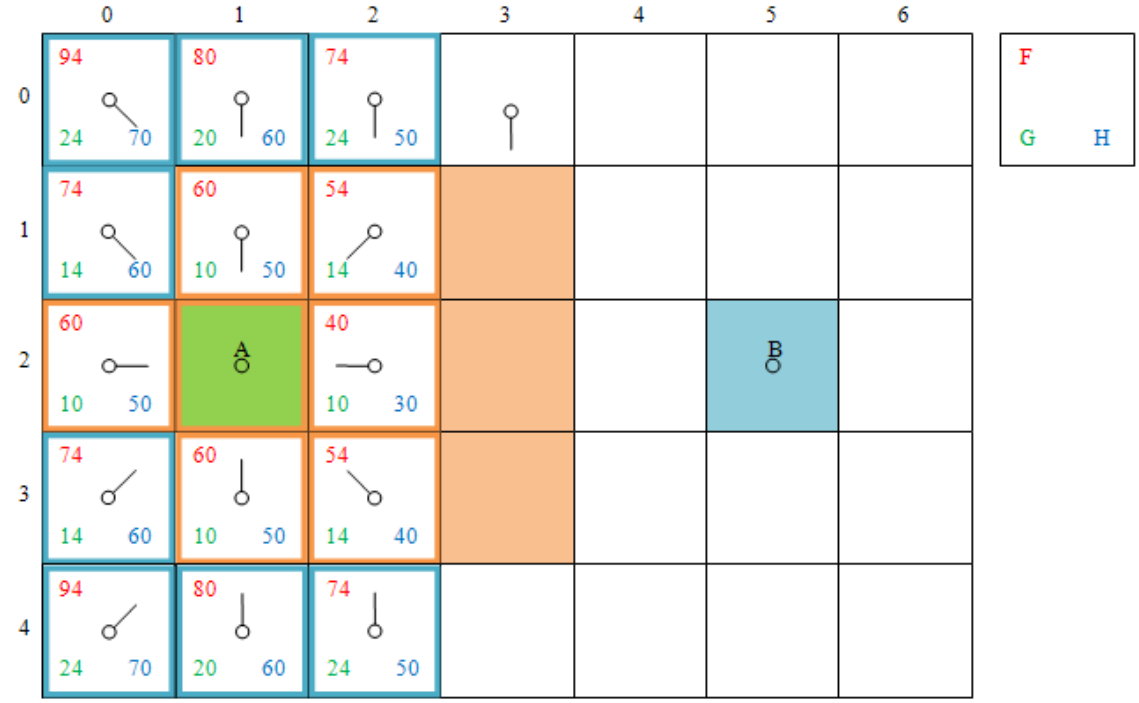
12. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [3,1]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



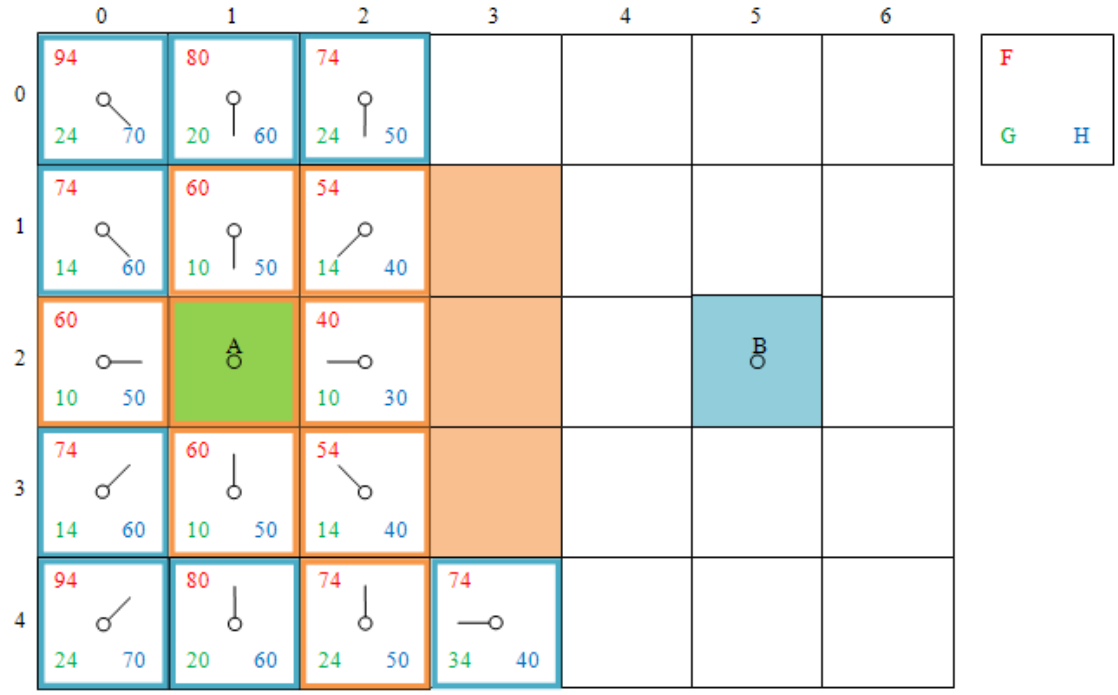
13. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [2,0]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



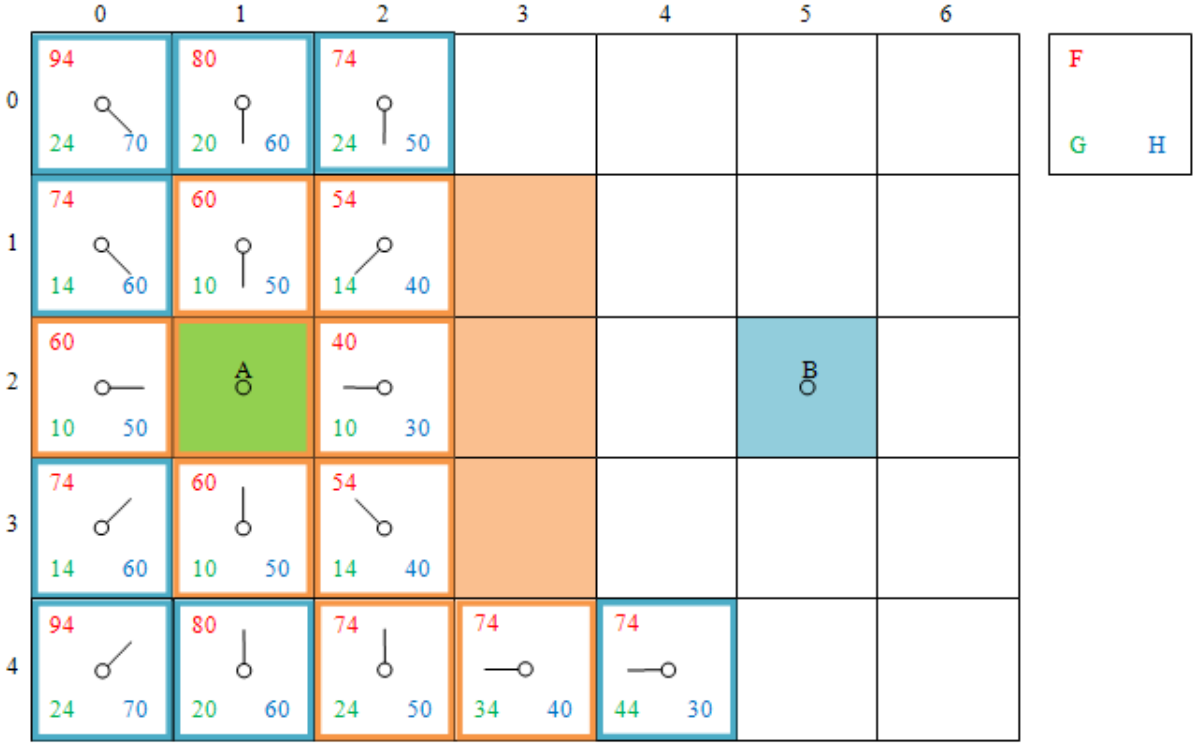
14. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [1,1]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



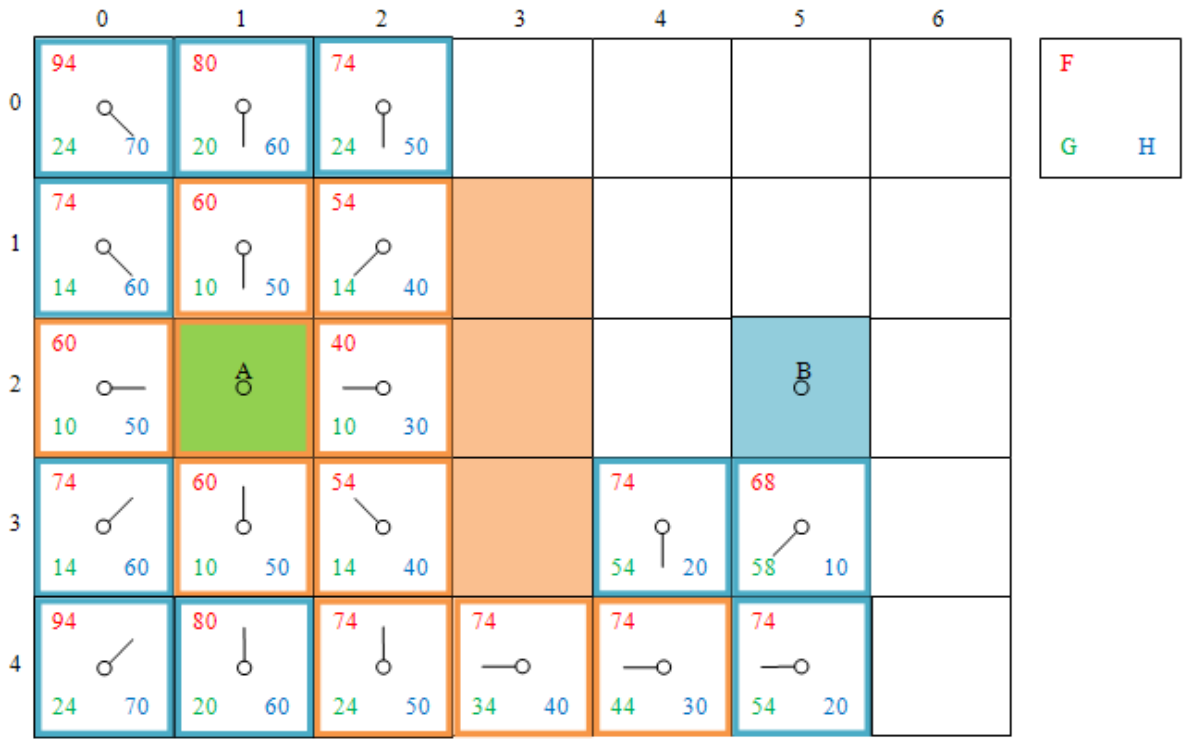
15. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [4,2]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



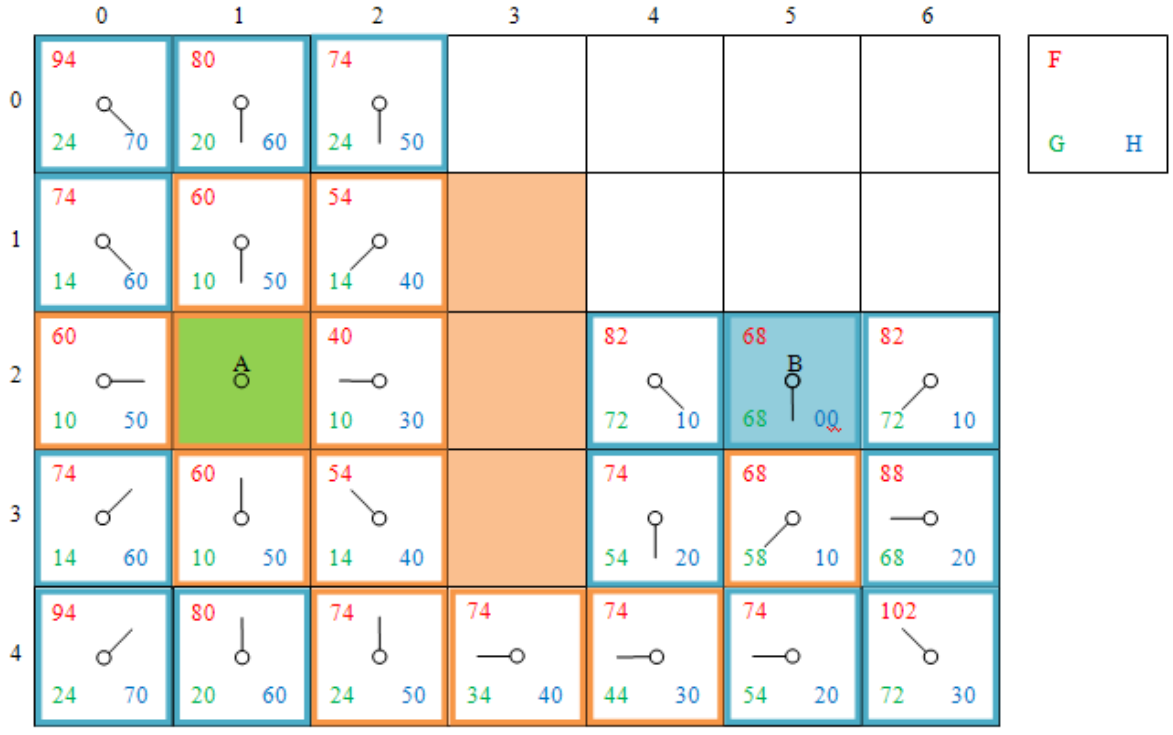
16. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [4,3]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



17. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [4,4]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.

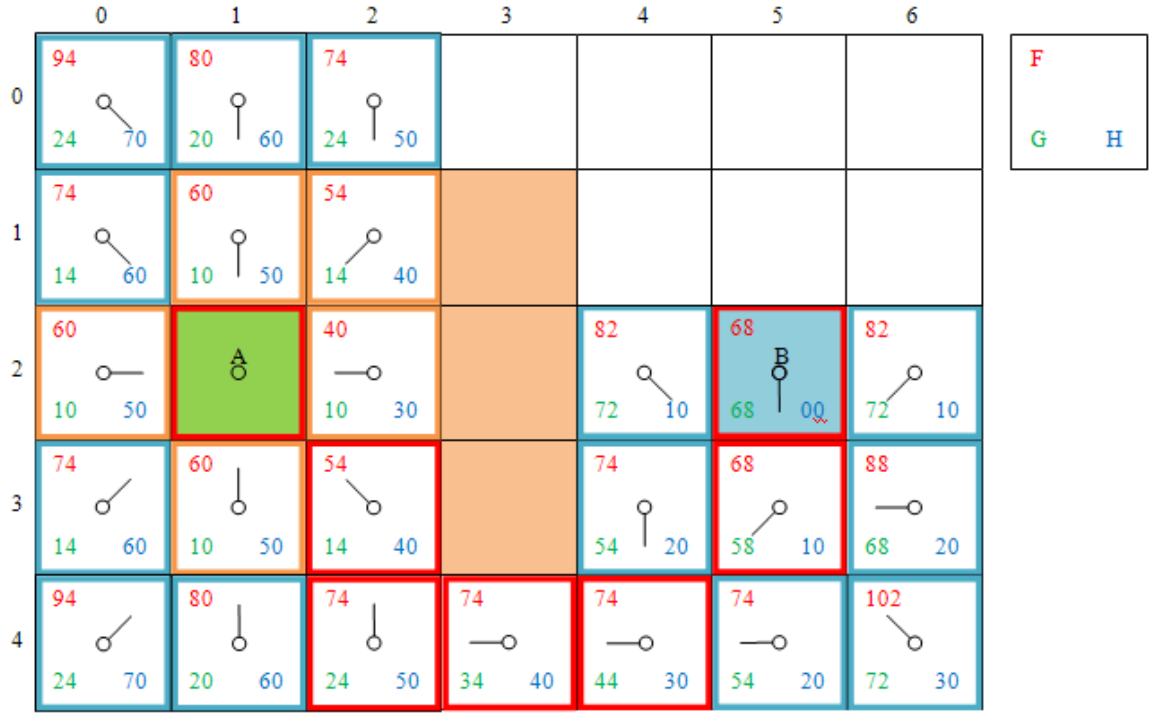


18. Обираємо наступну клітинку для відкритого списку – це клітина з координатами [3,5]. Видаляємо її з відкритого списку, додаємо у відкритий список її сусідів, визначаємо всі функції для них.



19. Цільова, тобто кінцева клітина знаходиться у відкритому списку, а це означає, що шлях від початкової до кінцевої точки знайдено. Тепер, відповідно до вказівників на батьківські клітинки можна пройти від кінцевої до початкової клітинки, а збережений шлях у зворотному напрямку – від початкової до кінцевої точкам – це і є найкоротший шлях. Алгоритм також припиняє

виконання, якщо список відкритих вершин – порожній, це означає, що не існує шляху від заданої точки до кінцевої вершини.



Зміст A* https://www.youtube.com/watch?v=eSOJ3ARN5FM
Зміст A* швидко https://www.youtube.com/watch?v=71CEj4gKDnE
A* https://www.youtube.com/watch?v=gCclsviUeUk
A* Алгоритмы Поиска Пути на Python. Алгоритм A*, Дейкстры, Поиск в ширину [Pygame]