Parallel A* search algorithm

Group 1

0716045 彭敘溶
0716067 何昱奇
07162 1 1 陳煜盛

Outline

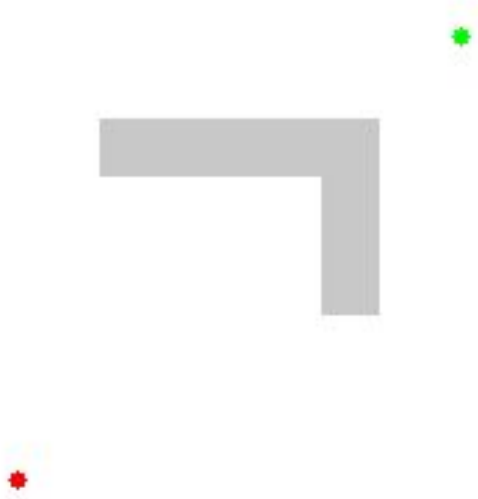
- ❑ A* search algorithm
- ❑ Experimental Setup
- ❑ Centralized Parallel A*
- ❑ Analyzing CPA
- ❑ Decentralized Parallel A*

A* search algorithm

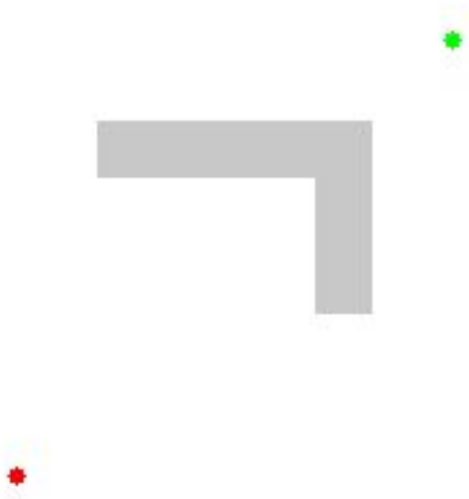
- Informed search algorithm
- Best first search
- Evaluation : $f(n) = g(n) + h(n)$
- $g(n)$: cost function, the actual cost from start node to node n
- $h(n)$: heuristic function
  - admissible => never overestimate
  - ex : In shortest path problem => stright line distance

A* search algorithm (Cont.)

- BFS



- A*



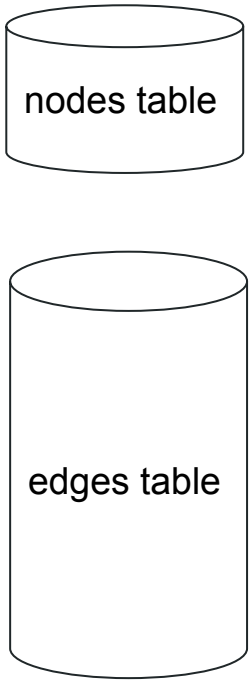
Experimental Setup

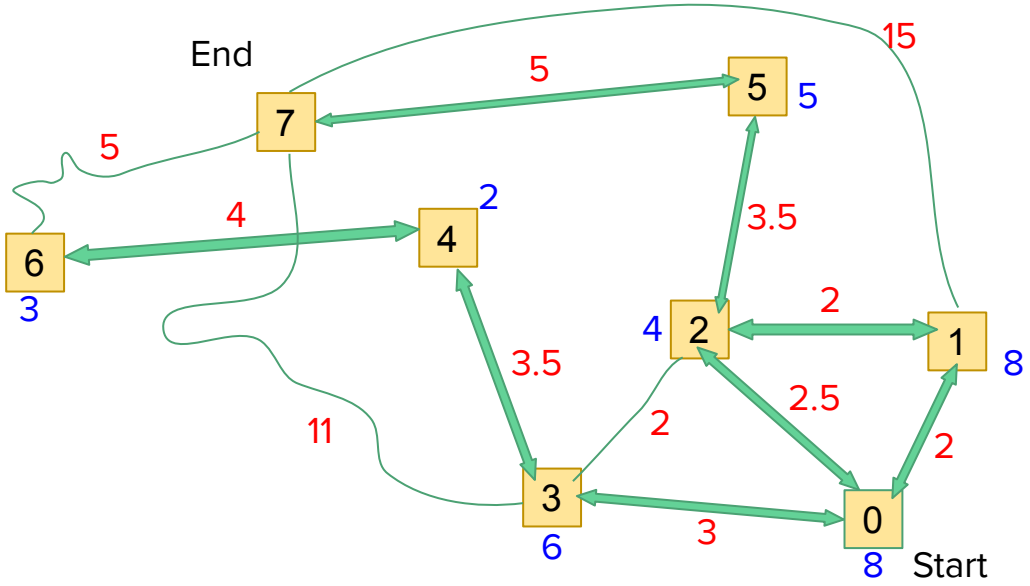
- Graph / Map Construction
  - Real road data from Hsinchu using OpenStreetMap
- Heuristic
  - The straight distance between each node and the target node

Centralized Parallel A*

- node
  - ID
  - g
  - h
  - previous
  - open
  - closed
  - Lock
- edge
  - start
  - end
  - distance
- Lock
  - Lock1 (OPEN/CLOSED)
  - Lock2(Incumbent cost)
- Node Set
  - OPEN
  - CLOSED

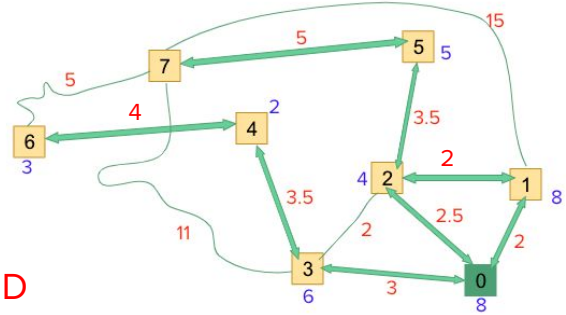
Centralized Parallel A* – An example





- edge distance
- heuristic

Centralized Parallel A*

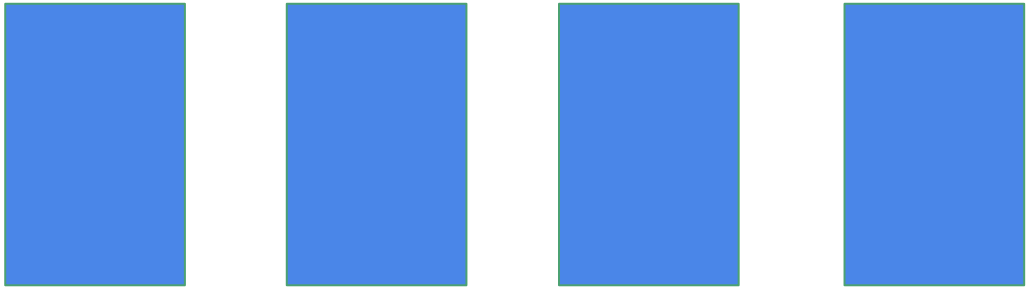


Start

OPEN

0

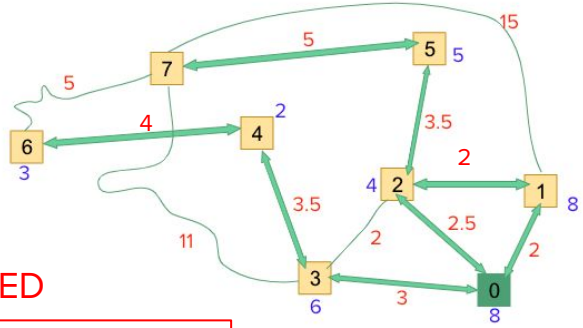
CLOSED



thread 0 thread 1 thread 2 thread 3

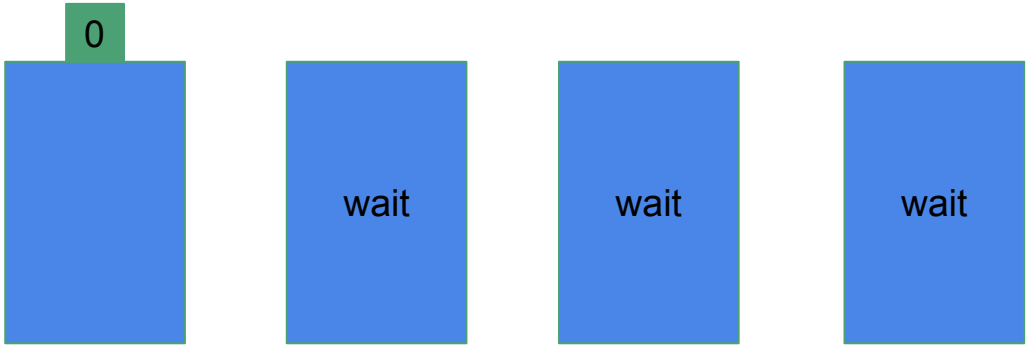
incumbent_cost = ∞

Centralized Parallel A*



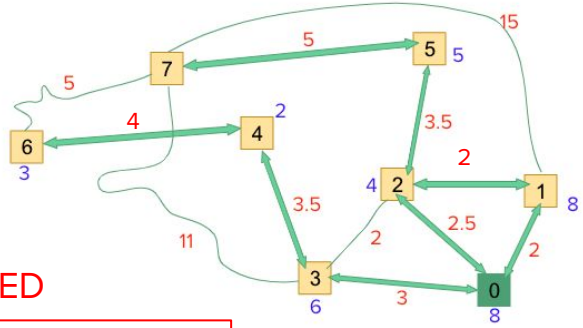
OPEN

CLOSED



incumbent_cost = ∞

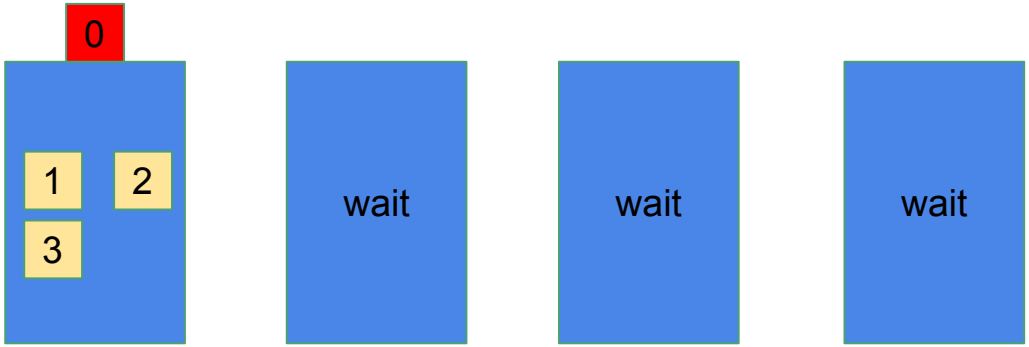
Centralized Parallel A*



OPEN

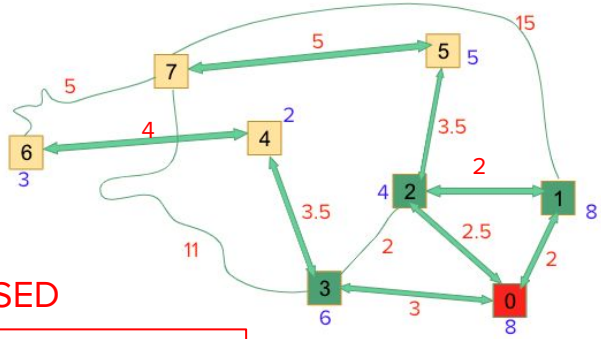
CLOSED

0



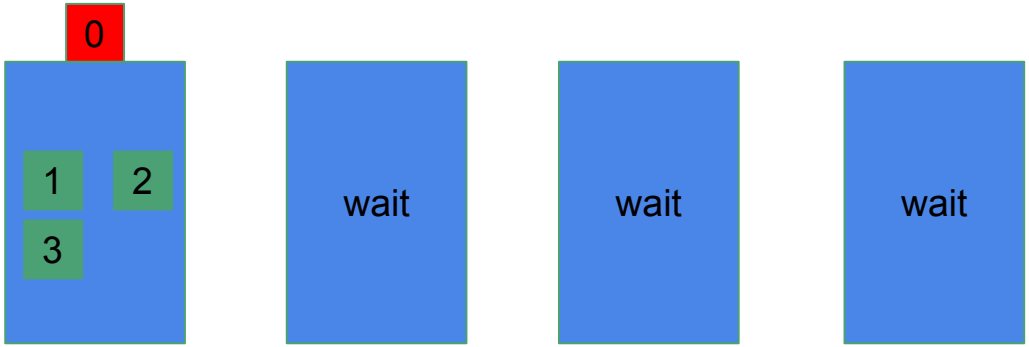
incumbent_cost = ∞

Centralized Parallel A*



OPEN

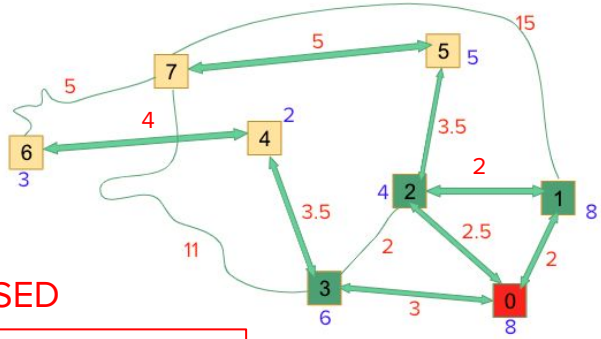
CLOSED



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

Centralized Parallel A*

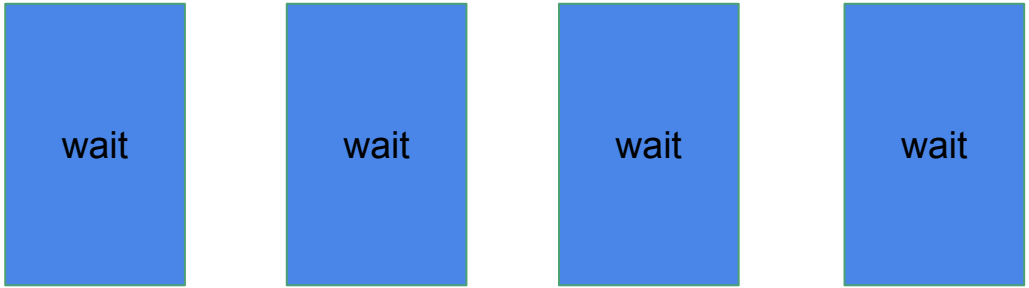


OPEN

finding best

CLOSED

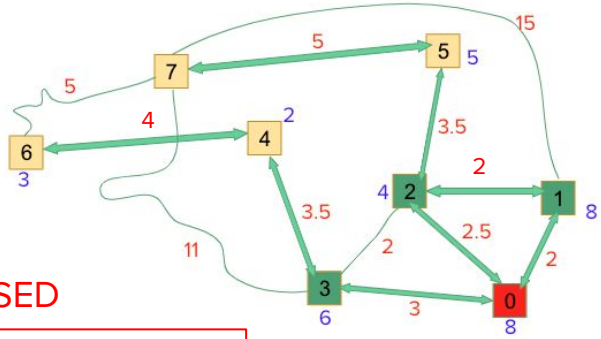
Lock1



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

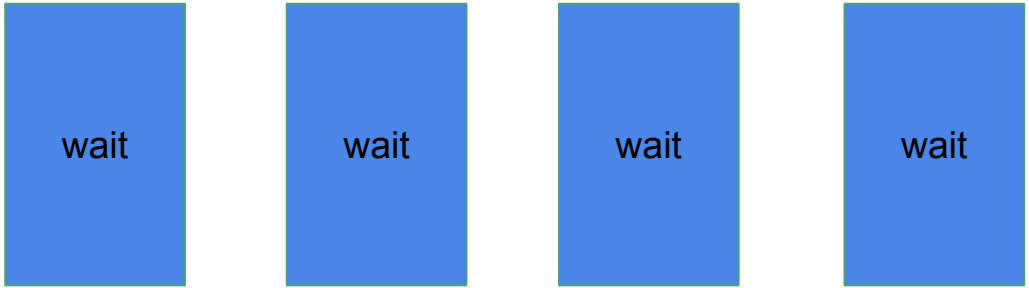
Centralized Parallel A*



OPEN

CLOSED

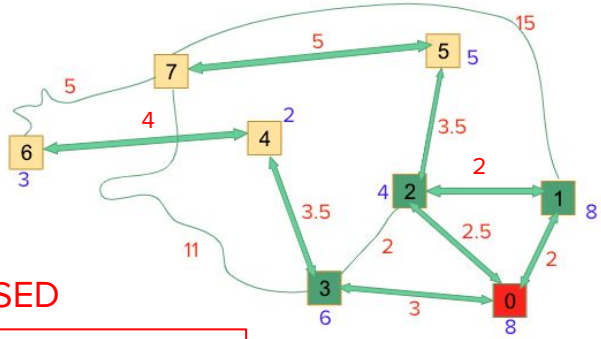
found



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

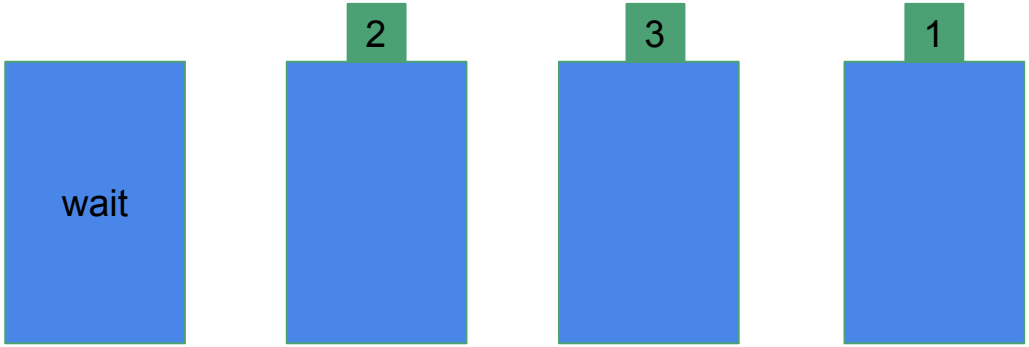
Centralized Parallel A*



OPEN

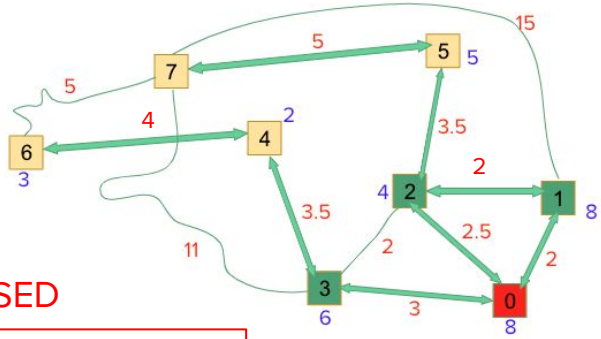
CLOSED

0



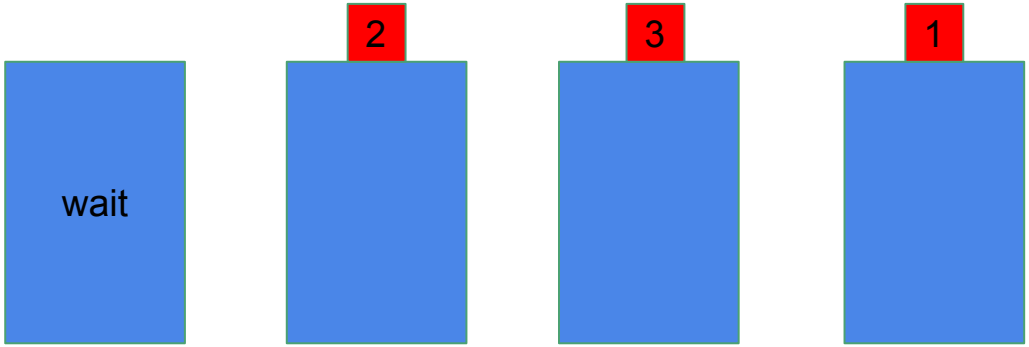
incumbent_cost = ∞

Centralized Parallel A*



OPEN

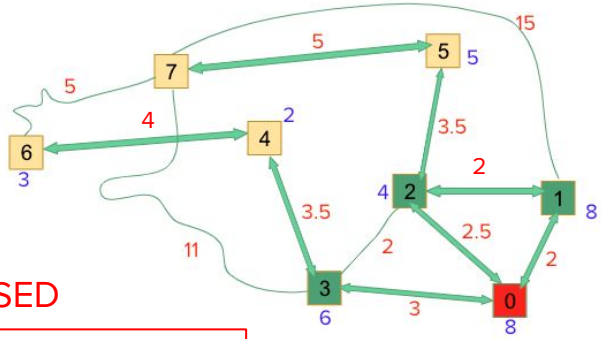
CLOSED



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

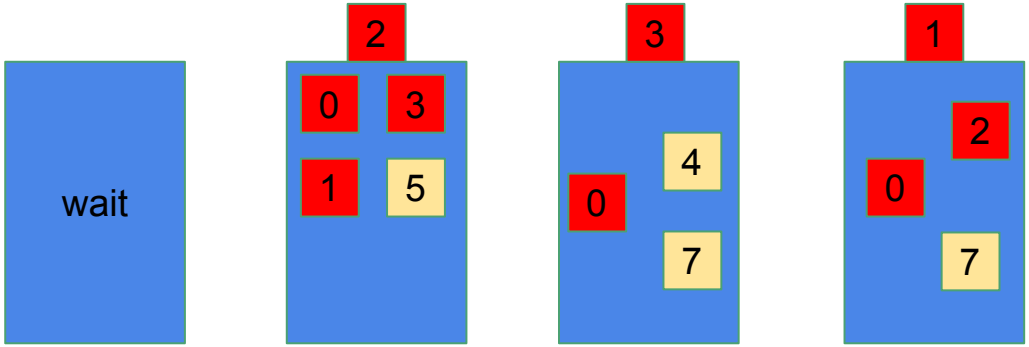
Centralized Parallel A*



OPEN

CLOSED

0 1 2 3

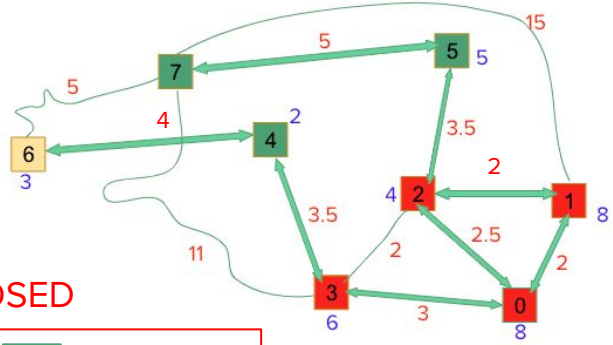


thread 0 thread 1 thread 2 thread 3

Node 1,2,3
Ambiguous
But no problem
=> finally closed

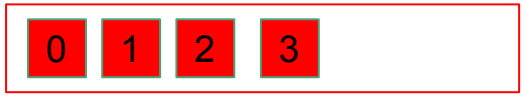
incumbent_cost = ∞

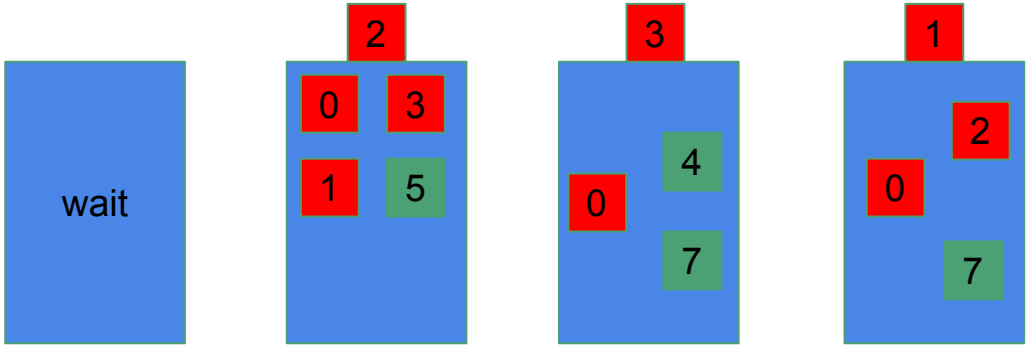
Centralized Parallel A*



OPEN

CLOSED



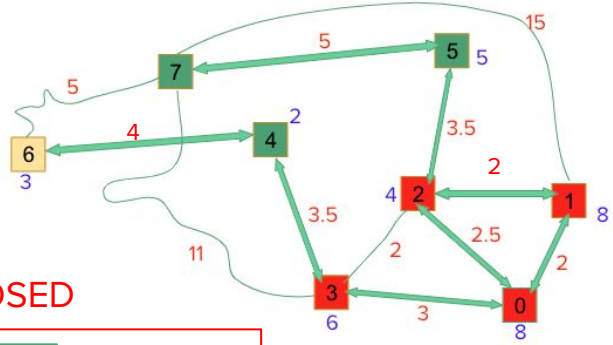


thread 0 thread 1 thread 2 thread 3

node 7 is dangerous.
lock it for safety.

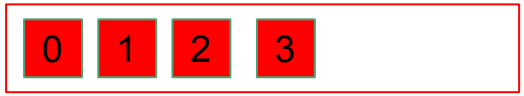
incumbent_cost = ∞

Centralized Parallel A*

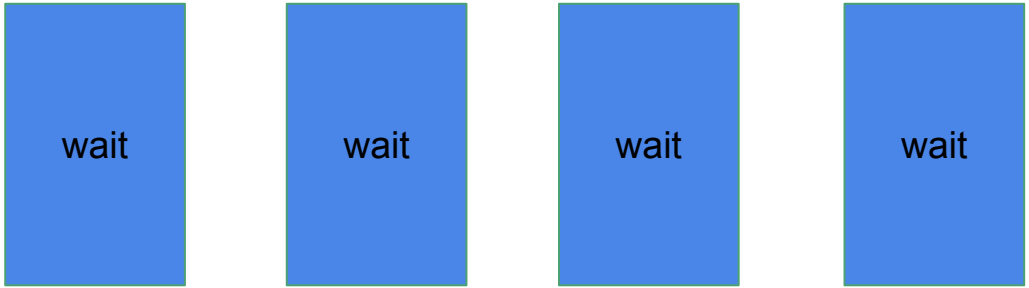


OPEN

CLOSED



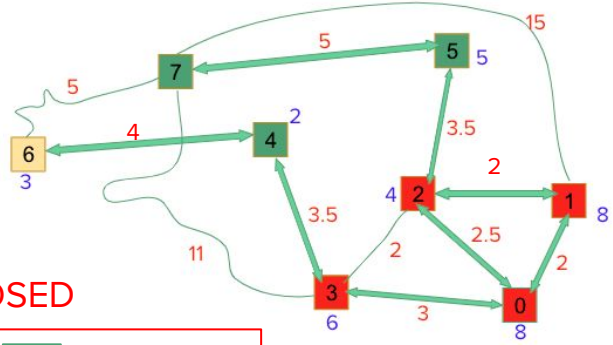
Lock1



thread 0 thread 1 thread 2 thread 3

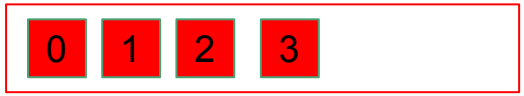
incumbent_cost = ∞

Centralized Parallel A*



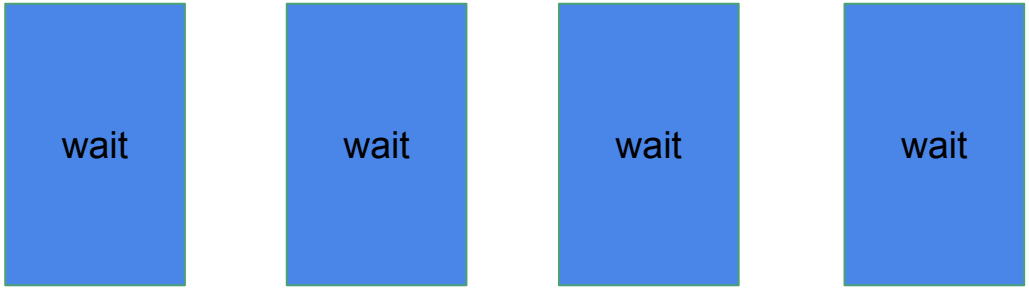
OPEN

CLOSED



finding best

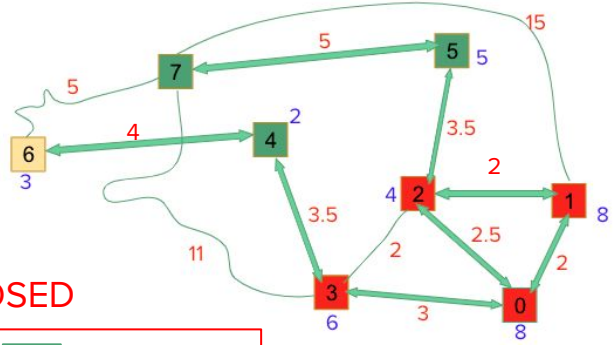
Lock1



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

Centralized Parallel A*



OPEN

4	5	7

CLOSED

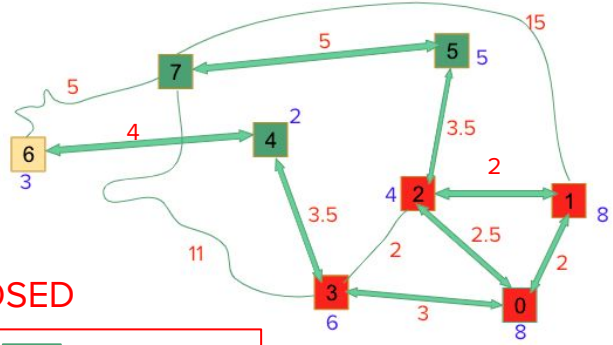
0	1	2	3

found

wait	wait	wait	wait
thread 0	thread 1	thread 2	thread 3

incumbent_cost = ∞

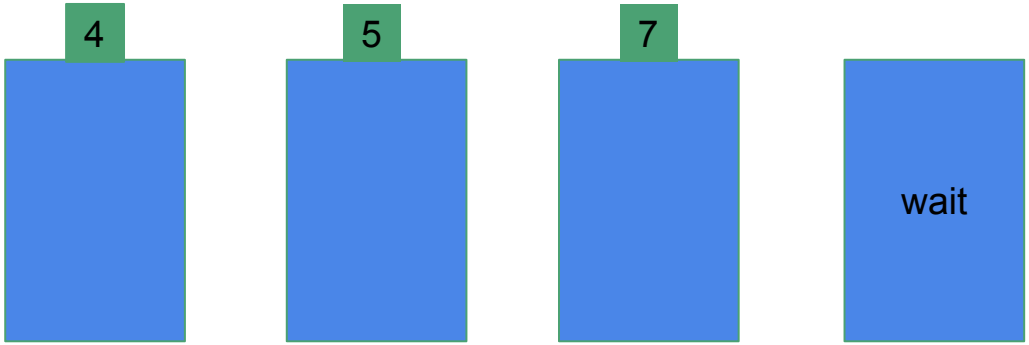
Centralized Parallel A*



OPEN

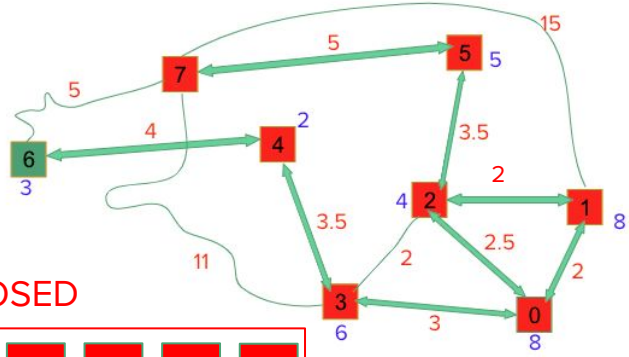
CLOSED

0 1 2 3



incumbent_cost = ∞

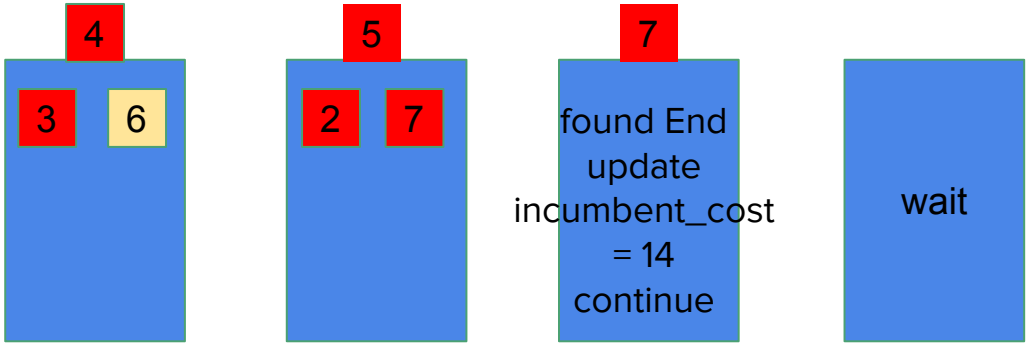
Centralized Parallel A*



OPEN

CLOSED

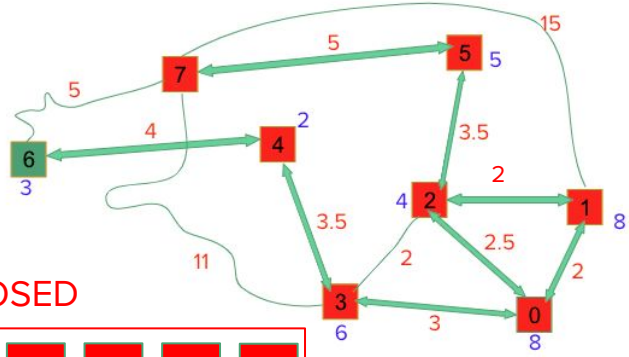
0 1 2 3 4 5 7



thread 0 thread 1 thread 2 thread 3

incumbent_cost = ∞

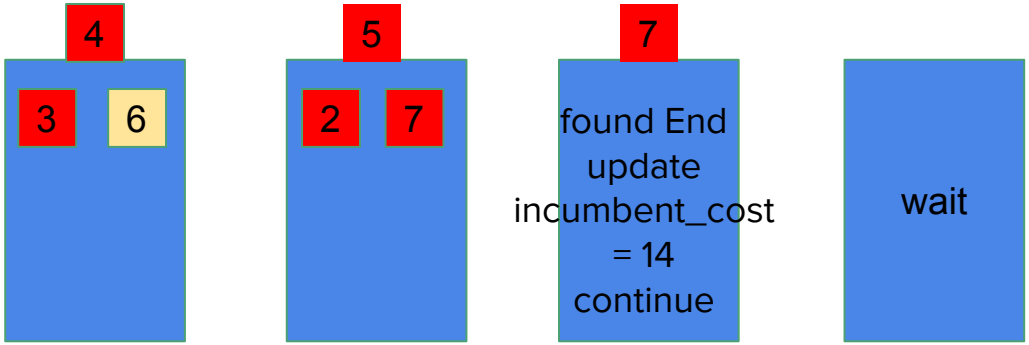
Centralized Parallel A*



OPEN

CLOSED

0 1 2 3 4 5 7

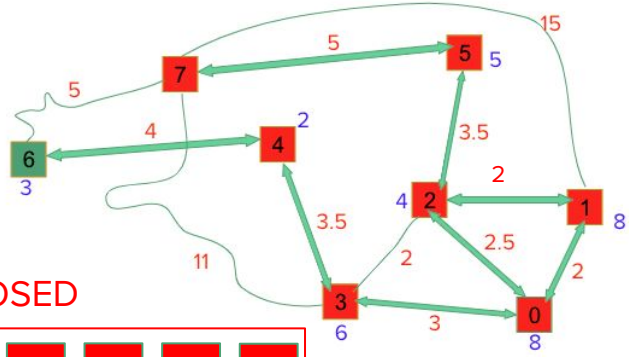


thread 0 thread 1 thread 2 thread 3

Lock2

incumbent_cost = ∞

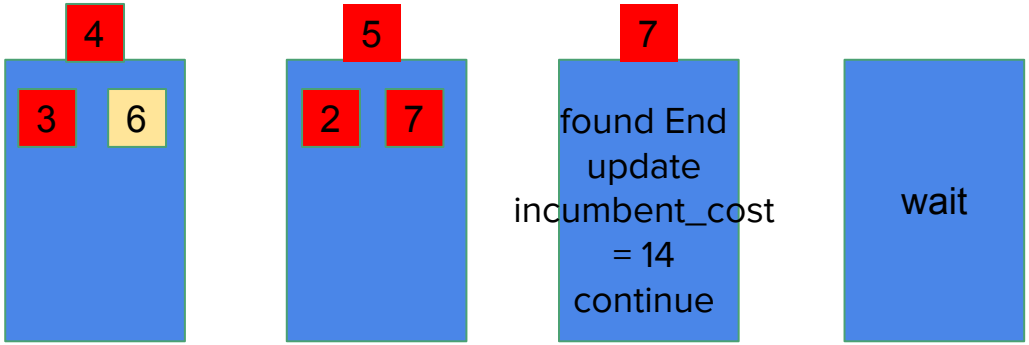
Centralized Parallel A*



OPEN

CLOSED

0 1 2 3 4 5 7

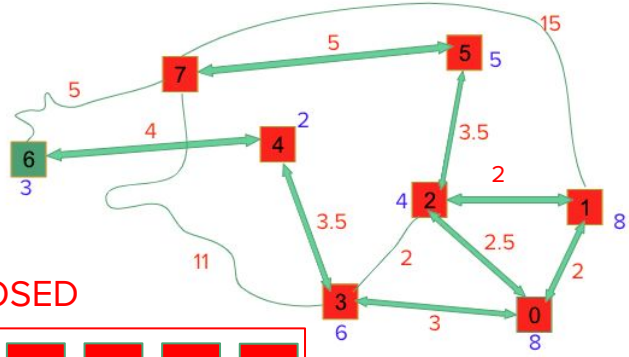


thread 0 thread 1 thread 2 thread 3

Lock2

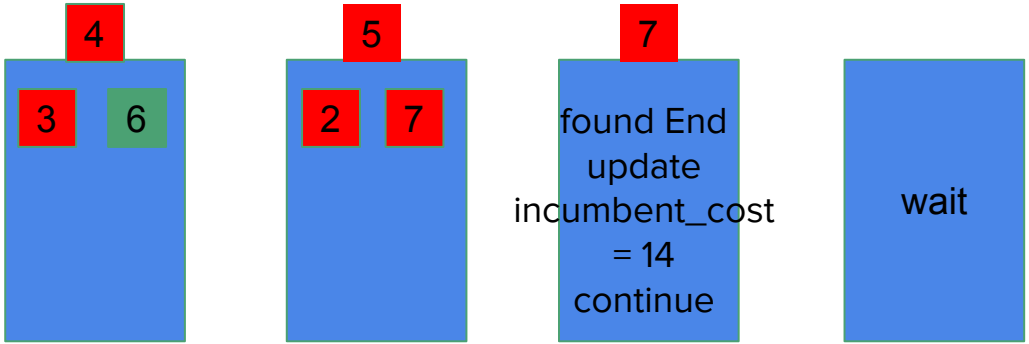
incumbent_cost = 14

Centralized Parallel A*



OPEN

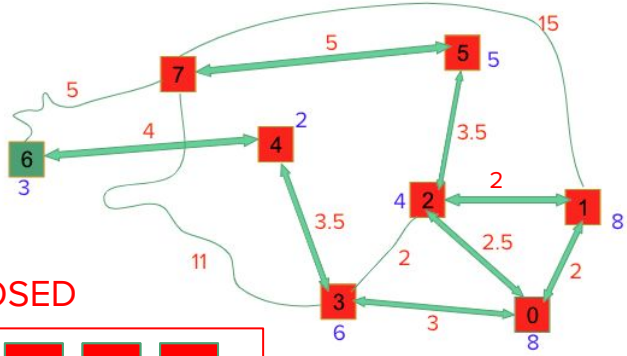
CLOSED



thread 0 thread 1 thread 2 thread 3

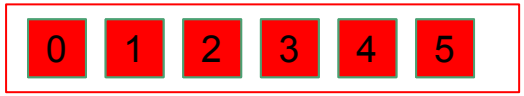
incumbent_cost = 14

Centralized Parallel A*

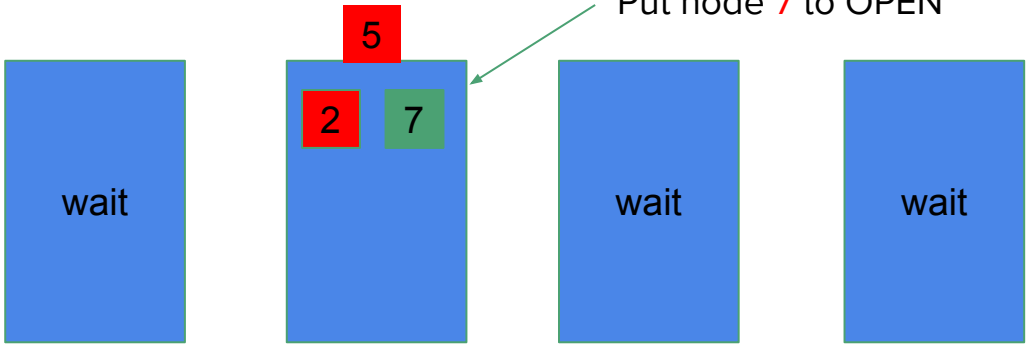


OPEN

CLOSED



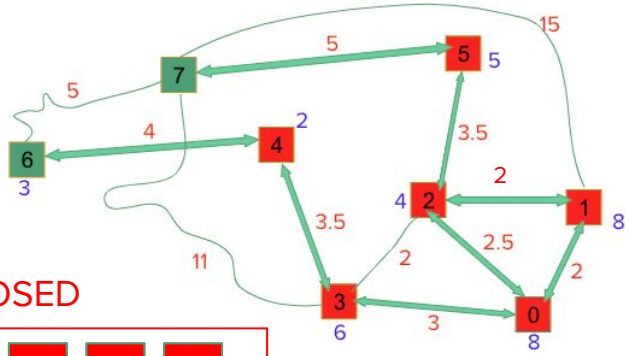
g(node 5) + edge(5,7) < g(node 7)
Put node 7 to OPEN



thread 0 thread 1 thread 2 thread 3

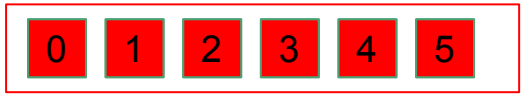
incumbent_cost = 14

Centralized Parallel A*



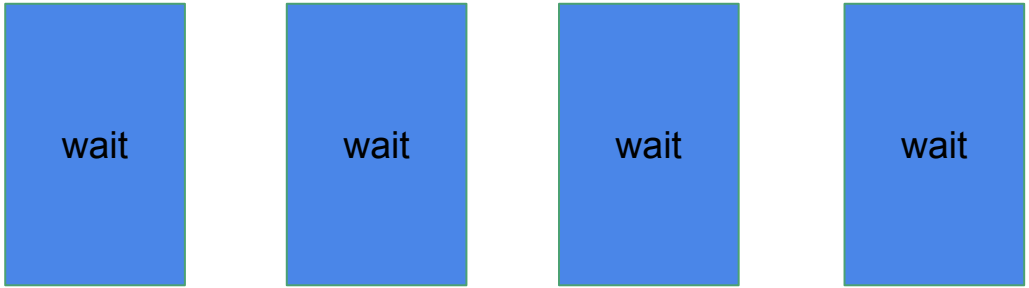
OPEN

CLOSED



finding best

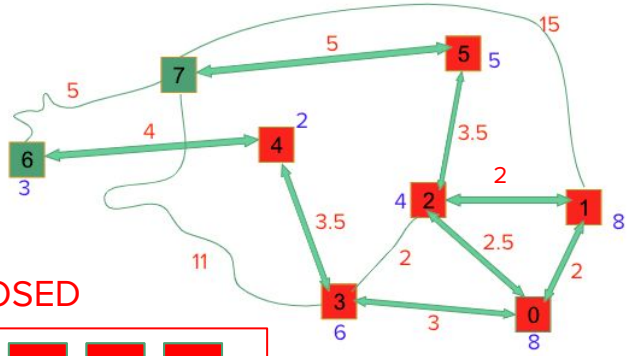
Lock 1



thread 0 thread 1 thread 2 thread 3

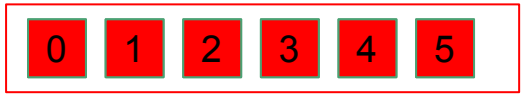
incumbent_cost = 14

Centralized Parallel A*

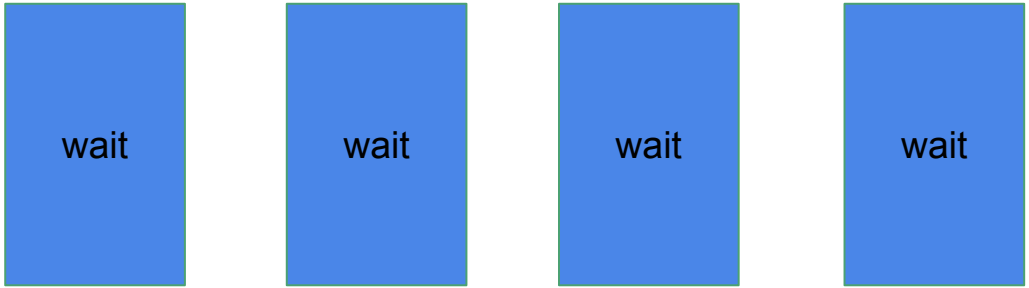


OPEN

CLOSED



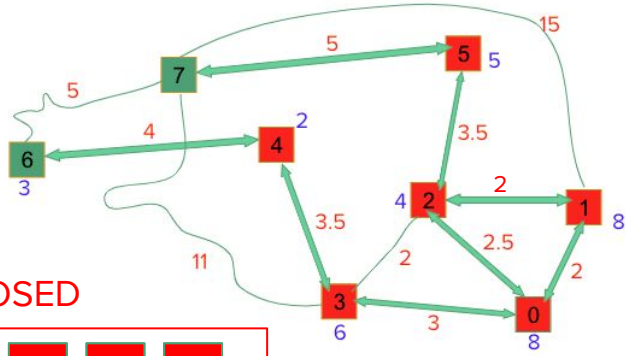
found



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 14

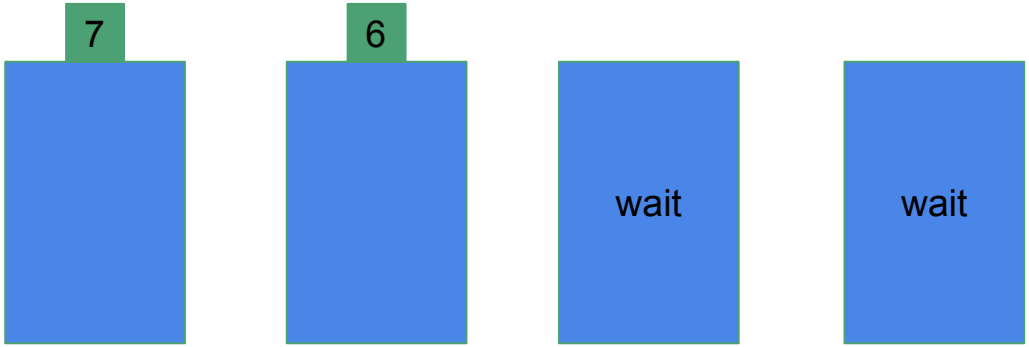
Centralized Parallel A*



OPEN

CLOSED

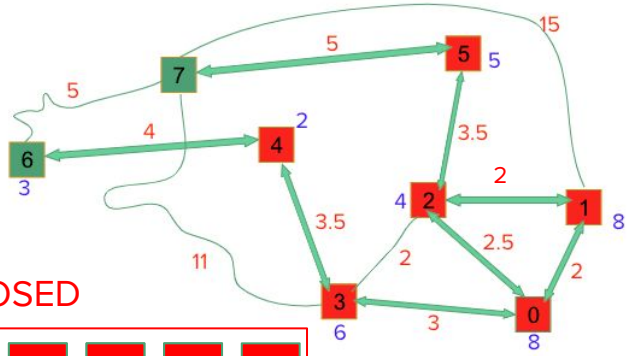
0 1 2 3 4 5



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 14

Centralized Parallel A*

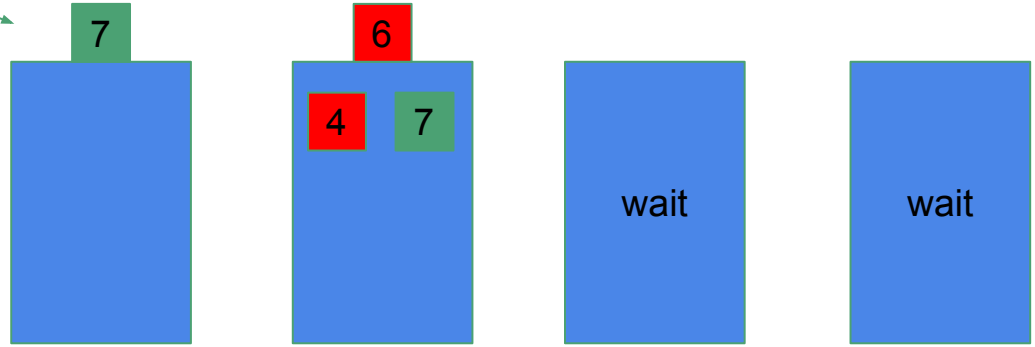


OPEN

CLOSED

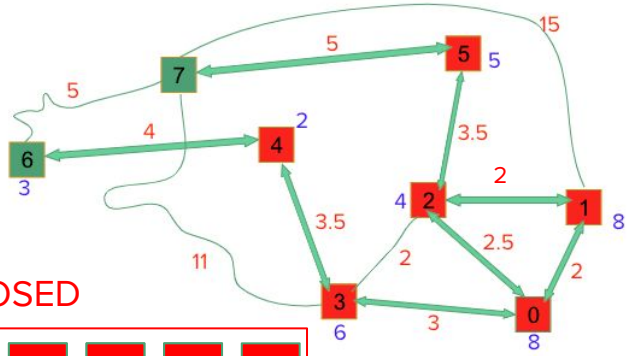
0 1 2 3 4 5 6

g(End) = 11 < incumbent_cost
remain OPEN



incumbent_cost = 14

Centralized Parallel A*

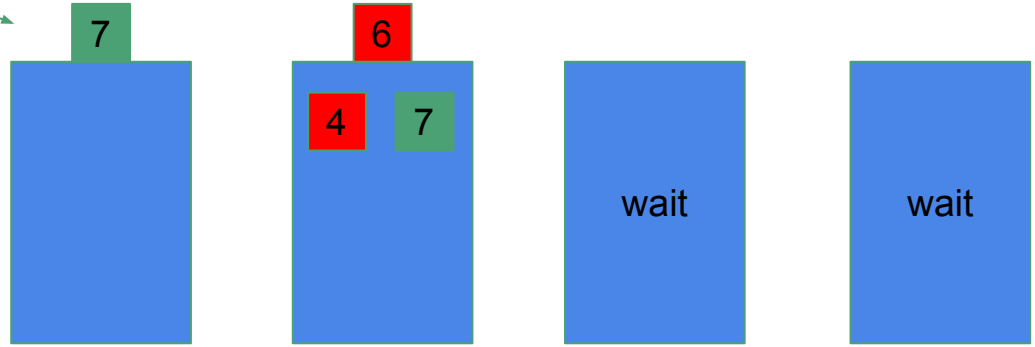


OPEN

CLOSED

0 1 2 3 4 5 6

g(End) = 11 < incumbent_cost
remain OPEN

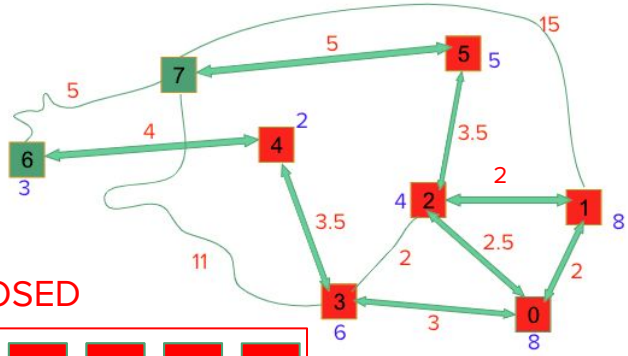


thread 0 thread 1 thread 2 thread 3

Lock2

incumbent_cost = 14

Centralized Parallel A*

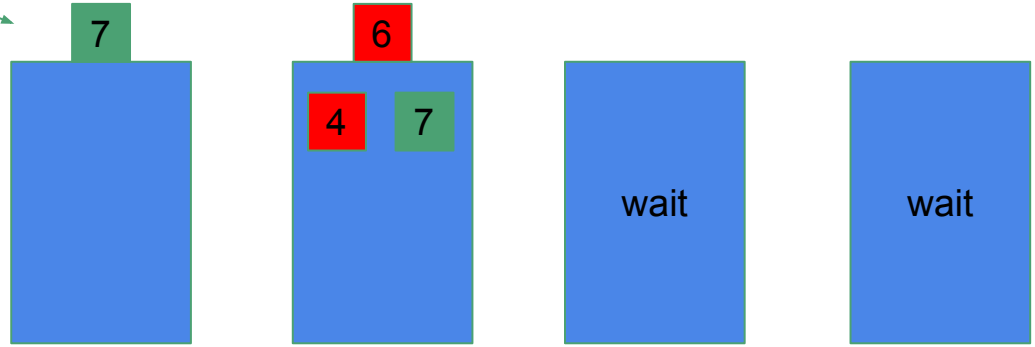


OPEN

CLOSED

0 1 2 3 4 5 6

g(End) = 11 < incumbent_cost
remain OPEN

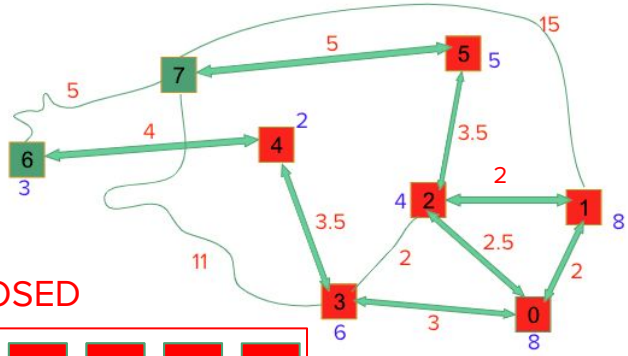


thread 0 thread 1 thread 2 thread 3

Lock2

incumbent_cost = 11

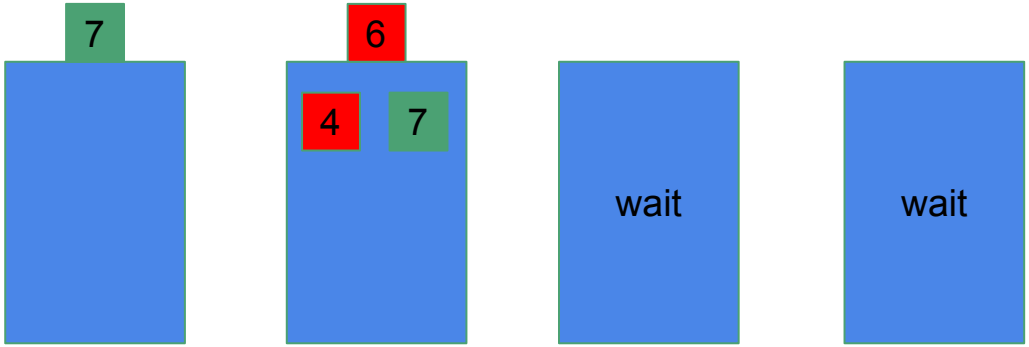
Centralized Parallel A*



OPEN

CLOSED

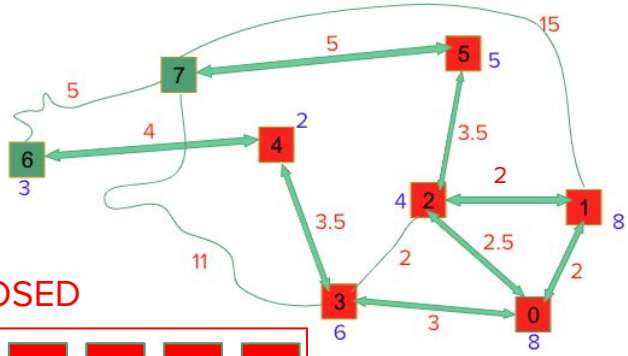
0 1 2 3 4 5 6



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

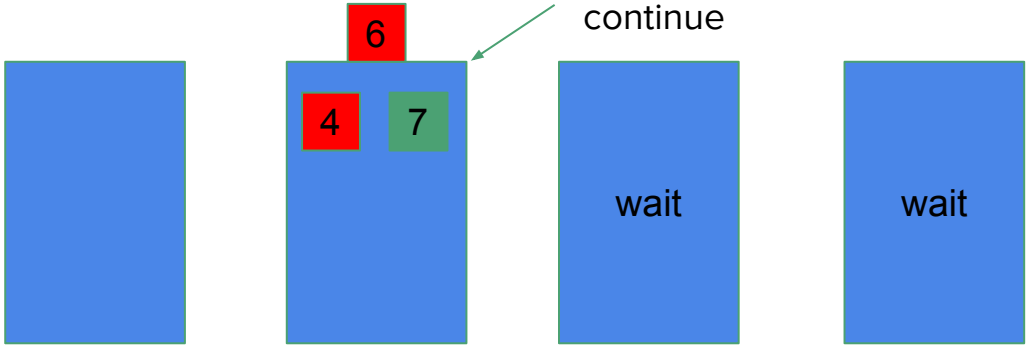
Centralized Parallel A*



OPEN

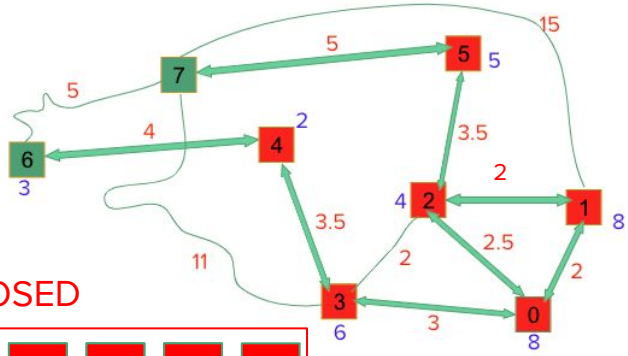
CLOSED

g(6) + edge(6,7) > g(7)
continue



incumbent_cost = 11

Centralized Parallel A*

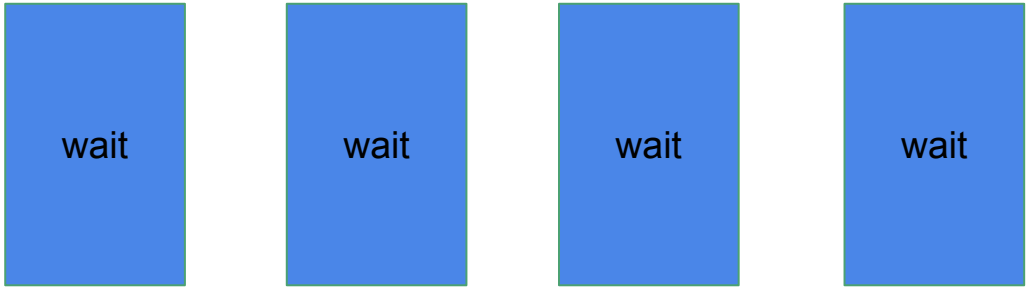


OPEN

CLOSED

finding best

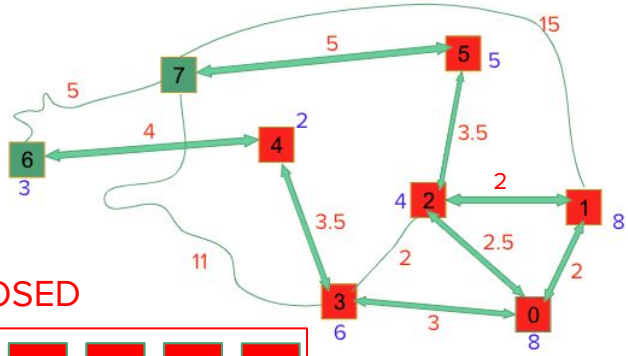
Lock1



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

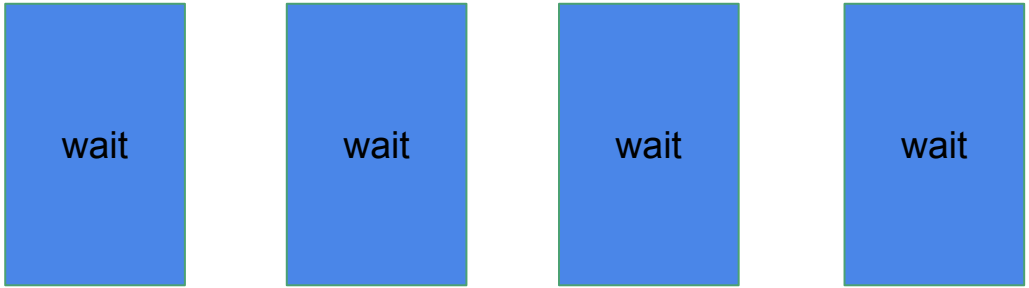
Centralized Parallel A*



OPEN

CLOSED

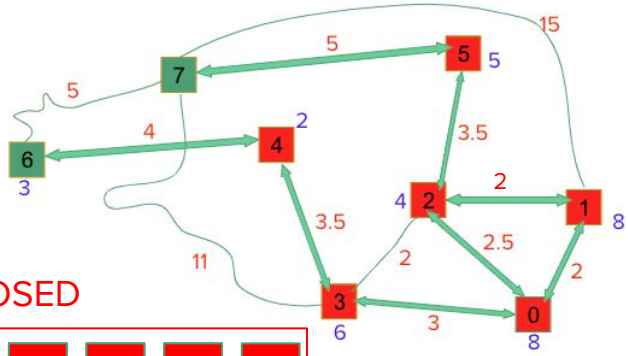
found



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

Centralized Parallel A*

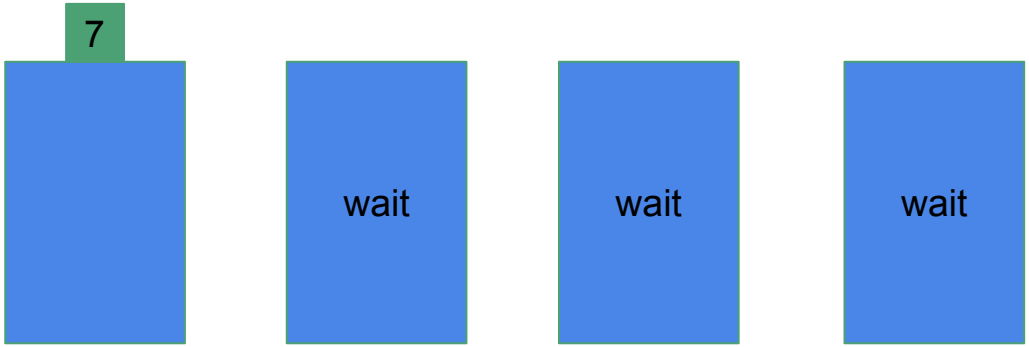


OPEN

CLOSED

0	1	2	3	4	5	6

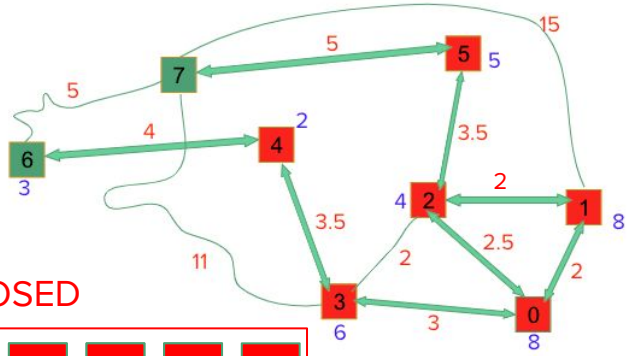
found



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

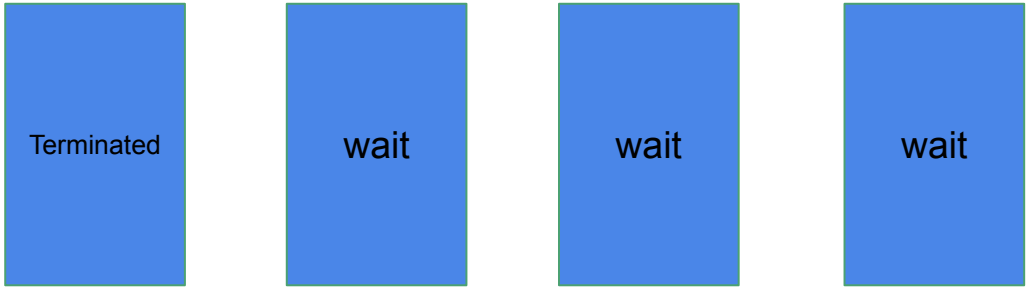
Centralized Parallel A*



OPEN

CLOSED

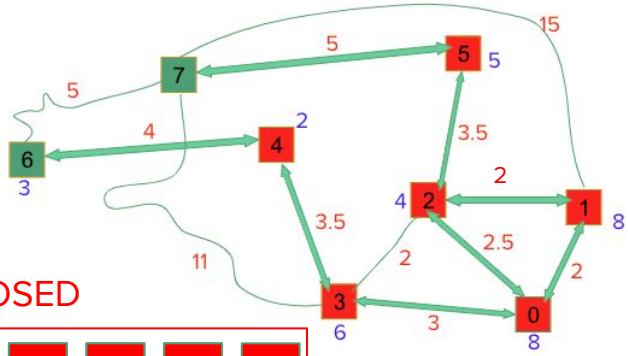
found



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

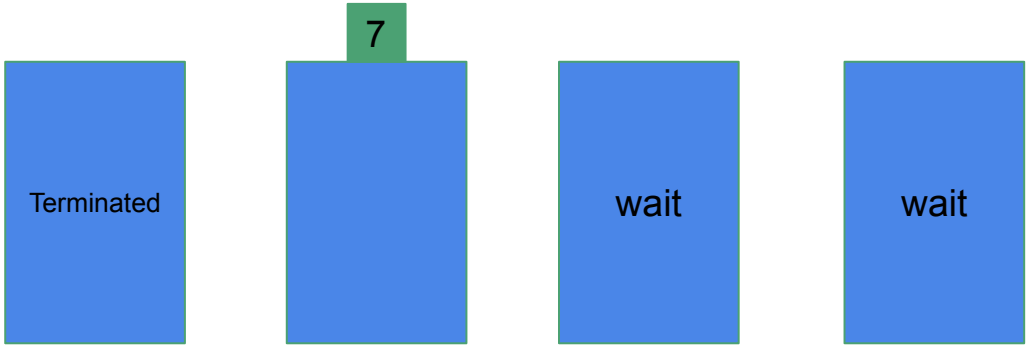
Centralized Parallel A*



OPEN

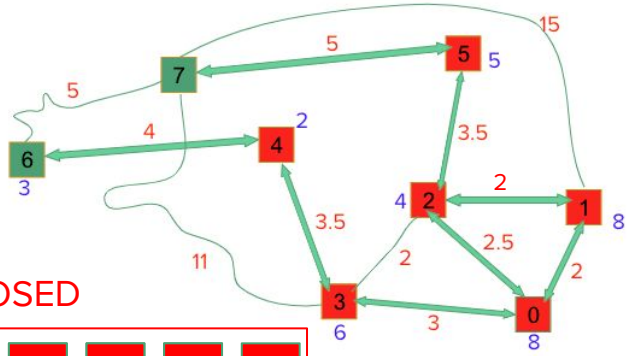
CLOSED

0	1	2	3	4	5	6



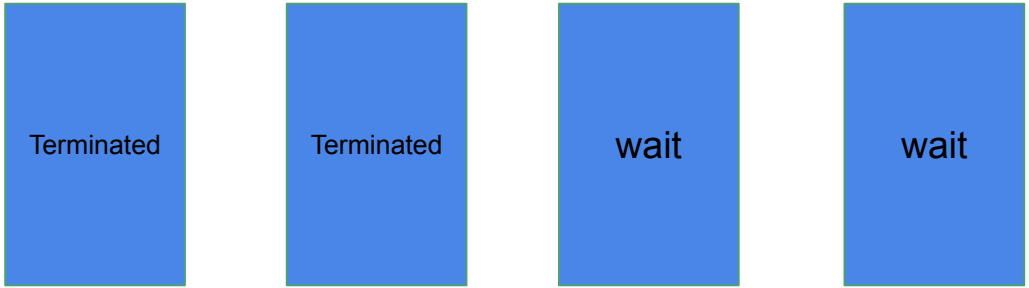
incumbent_cost = 11

Centralized Parallel A*



OPEN

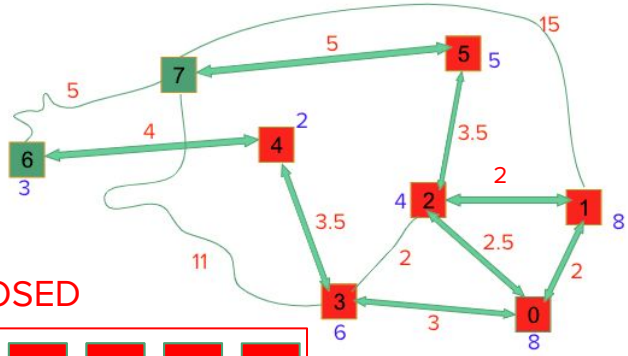
CLOSED



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

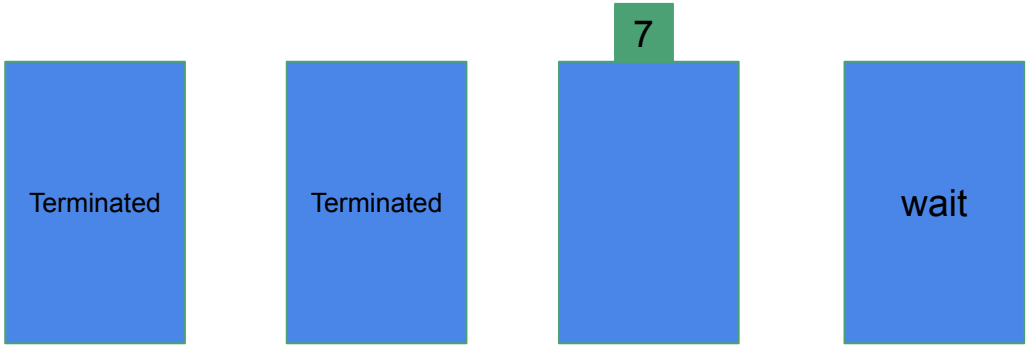
Centralized Parallel A*



OPEN

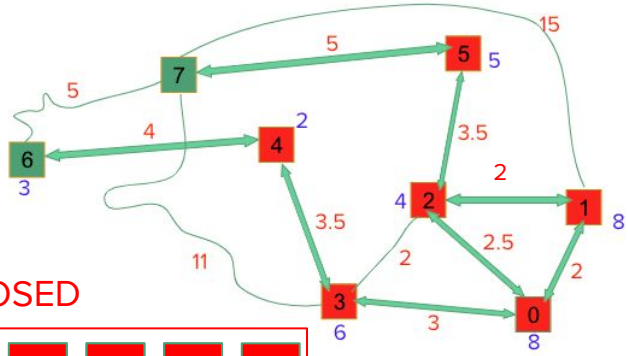
CLOSED

0	1	2	3	4	5	6



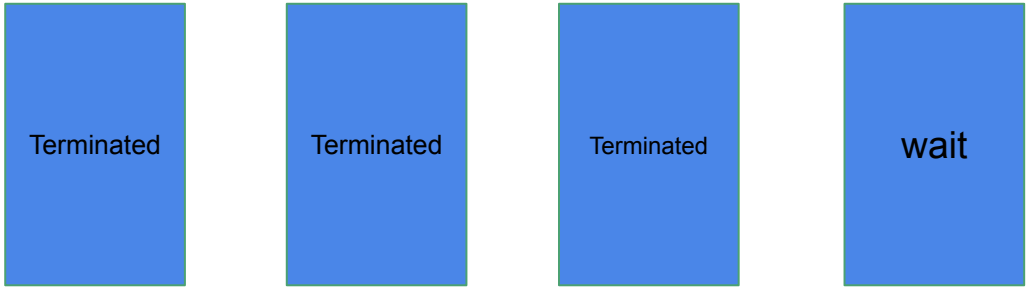
incumbent_cost = 11

Centralized Parallel A*



OPEN

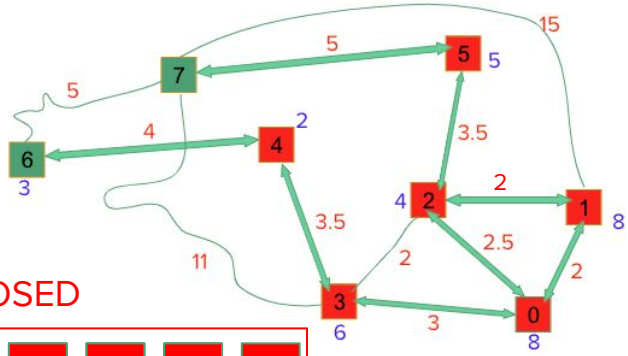
CLOSED



thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

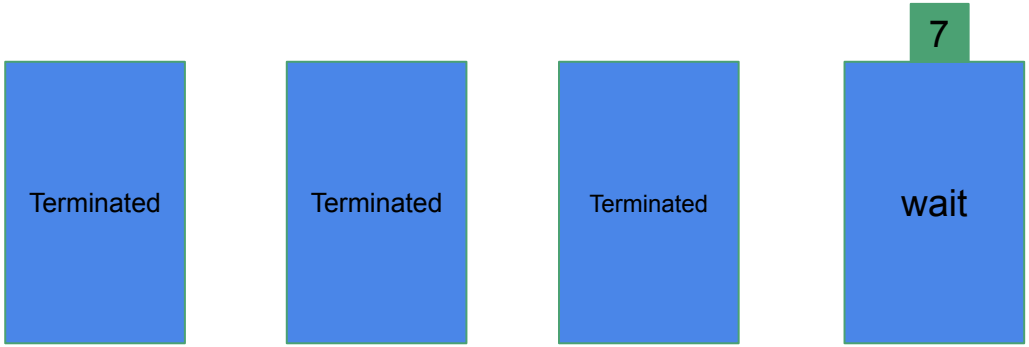
Centralized Parallel A*



OPEN

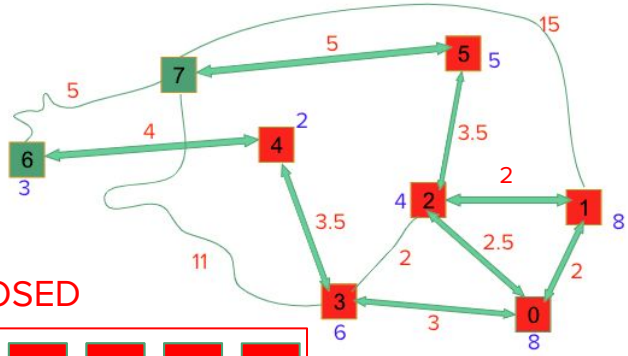
CLOSED

0 1 2 3 4 5 6



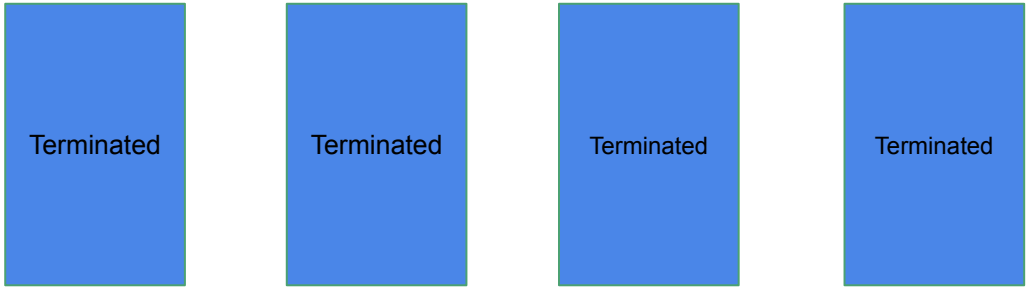
incumbent_cost = 11

Centralized Parallel A*



OPEN

CLOSED

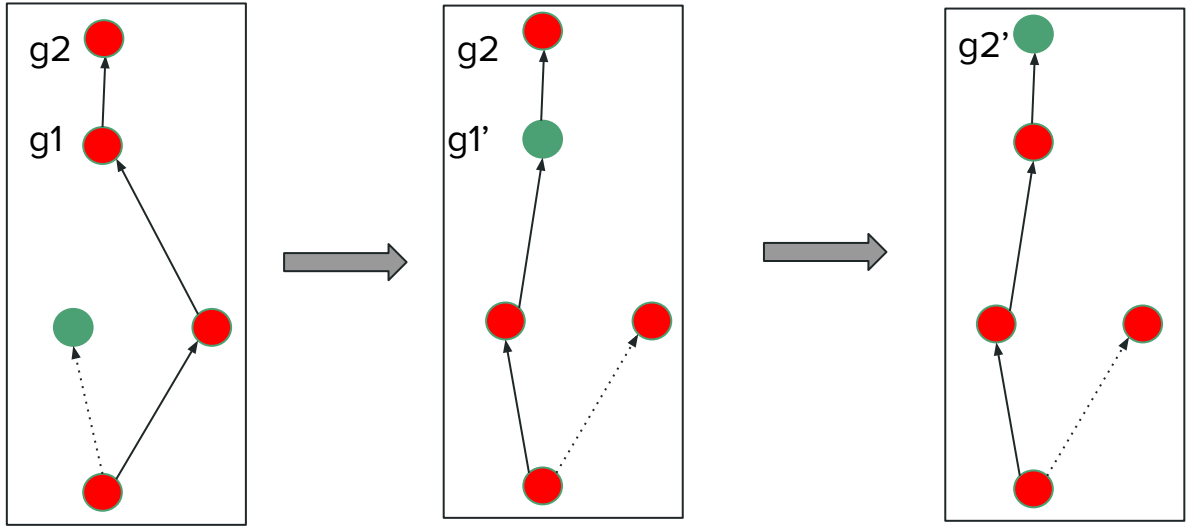


thread 0 thread 1 thread 2 thread 3

incumbent_cost = 11

Centralized Parallel A*

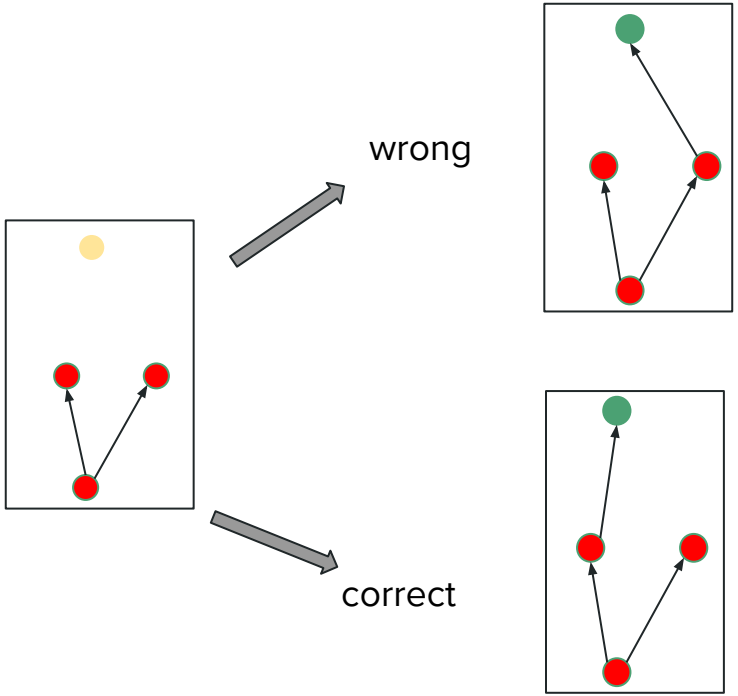
- Case 1



g1' < g1
g2' < g2

Centralized Parallel A*

- Case 2



Algorithm 2: Simple Parallel A* (SPA*)

```
Initialize OPEN_shared to {s_0};
Initialize Lock l_o, l_i;
Initialize incumbent.cost = ∞;
In parallel, on each thread, execute 5-32;
while TerminateDetection() do
    if OPEN_shared = ∅ or Smallest f(n) value of n ∈ OPEN_shared ≥ incumbent.cost then
        Continue;
    AcquireLock(l_o);
    Get and remove from OPEN_shared a node n with a smallest f(n);
    ReleaseLock(l_o);
    Add n to CLOSED_shared ;
    if n is a goal node then
        AcquireLock(l_i);
        if path cost from s_0 to n < incumbent.cost then
            incumbent = path from s_0 to n;
            incumbent.cost = path cost from s_0 to n;
        ReleaseLock(l_i);
    for every successor n' of n do
        g_1 = g(n) + c(n, n');
        if n' ∈ CLOSED_shared then
            if g_1 < g(n') then
                Remove n' from CLOSED_shared and add it to OPEN_shared ;
            else
                Continue;
        else
            if n' ∉ OPEN_shared then
                Add n' to OPEN_shared ;
            else if g_1 ≥ g(n') then
                Continue;
        Set g(n') = g_1;
        Set f(n') = g(n') + h(n');
        Set parent(n') = n;
if incumbent.cost = ∞ then
    Return failure (no path exists);
else
    Return solution path from s_0 to n;
```

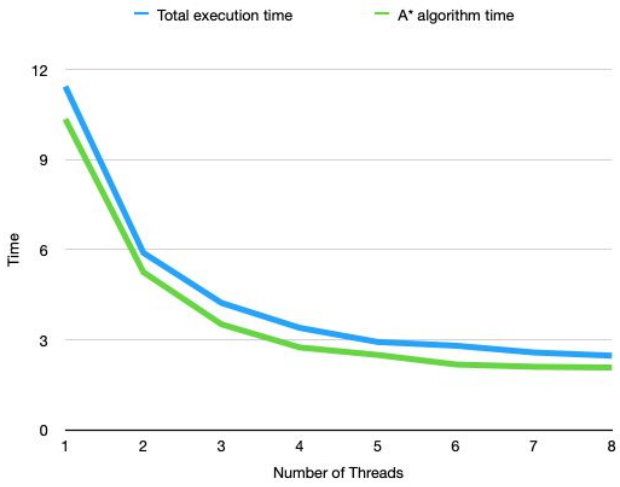
Lock

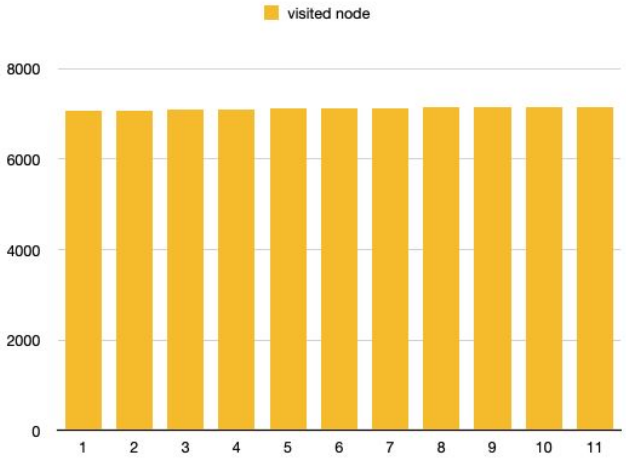
Unlock

Centralized Parallel A*

- Result

Enviroment : Macbook Pro 2019 13吋 CPU : i5- 8257U 4 Core 8 Thread			
Number of thread	Total execution time	A* algorithm time	visited node
1	11.433	10.347	7074
2	5.896	5.252	7080
3	4.228	3.516	7093
4	3.399	2.747	7104
5	2.928	2.496	7118
6	2.804	2.179	7130
7	2.579	2.103	7135
8	2.472	2.077	7137
9	2.601	2.088	7140
10	2.484	1.966	7147
11	2.555	1.987	7151





Centralized Parallel A*

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

- Amdahl's law
  - **p** : The proportion of parallelizable region.
  - **1 - p** : The proportion of non-parallelizable region.
  - **s = 8** : Number of threads.
  - $S_{\text{latency}}(s) =$ **4.98** : speed up

Number of thread	Total execution time	A* algorithm time
1	11.433	10.347
8	2.472	2.077

4.98x

  - => p = 0.91
  - Cost a lot when finding neighbor.

Centralized Parallel A*

- When finding neighbors.
  - Search edges table to get neighbors' ID.
  - Search nodes table to get neighbor 1's g, h, open, closed ...
  - Search nodes table to get neighbor 2's g, h, open, closed ...
  - Search nodes table to get neighbor 3's g, h, open, closed ...
  - ...

nodes table

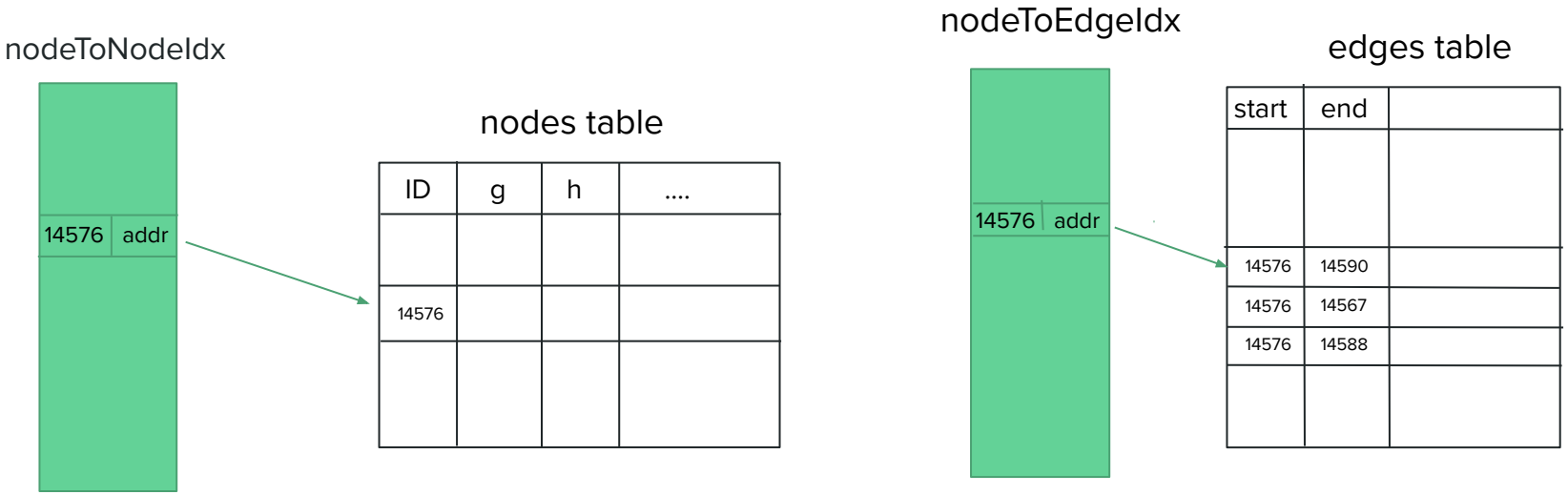
ID	g	h	....
14567			
...			
14576			
...			
14588			
...			
14590			

edges table

start	end	
14576	14590	
14576	14567	
14576	14588	

Centralized Parallel A*

- Solution : Use Index

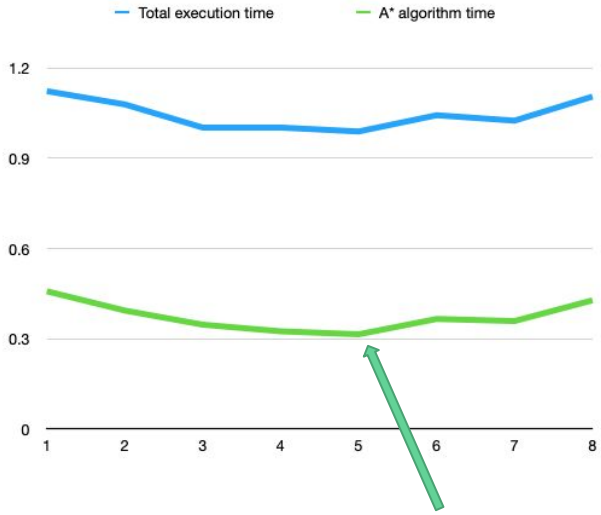


Centralized Parallel A*

Centralized parallel A* with new structure

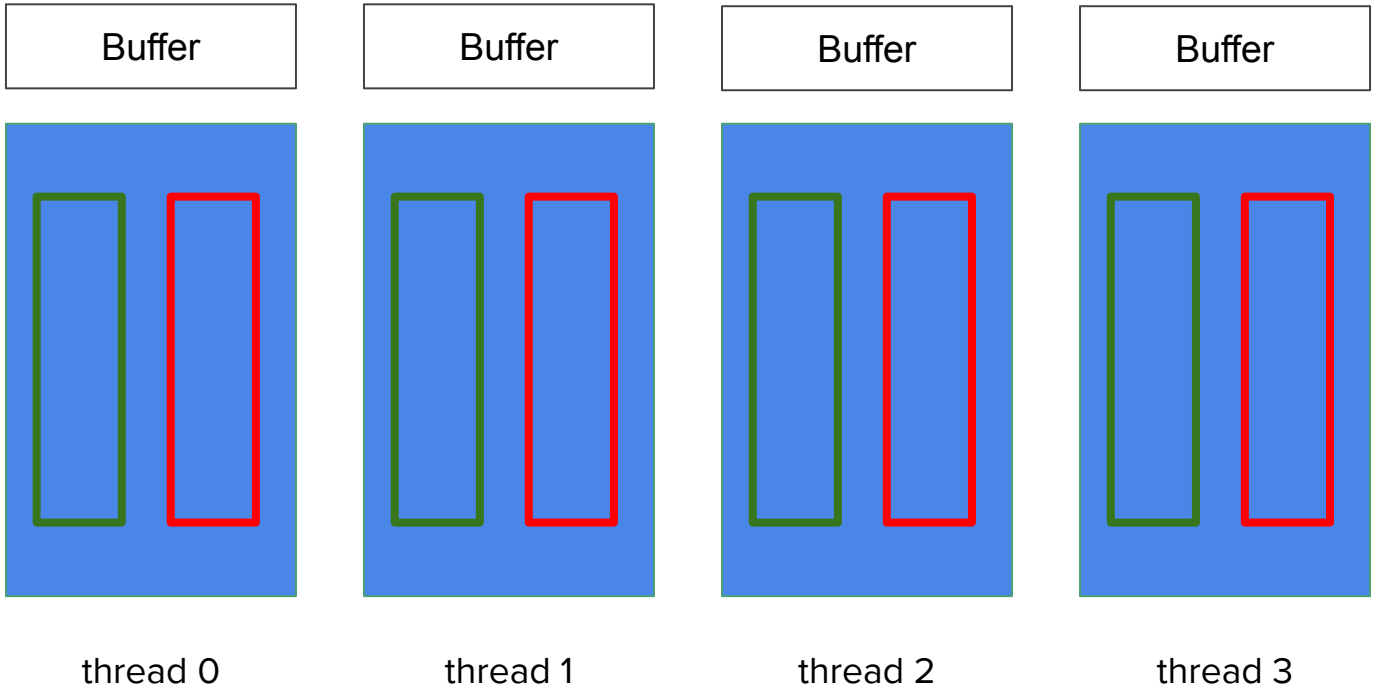
Number of thread	Total execution time	A* algorithm time	Visited node
1	1.123	0.458	7074
2	1.079	0.394	7075
3	1.002	0.347	7075
4	1.002	0.325	7074
5	0.989	0.315	7075
6	1.043	0.366	7075
7	1.025	0.359	7076
8	1.105	0.428	7075
9	2.601	2.088	7077
10	2.484	1.966	7077
11	2.555	1.987	7078

Intel i5 8257U is 4 Core 8 Thread



minimum

Decentralized Parallel A*



Decentralized Parallel A*

- Each thread has their own OPEN, CLOSED, and BUFFER set
- Faster than CPA
- Constraint: The Graph must not be bidirectional
  - Real world map is bidirectional
  - Oscillating between two nodes

References

[1] Alex Fukunaga, Adi Botea, Yuu Jinnai, and Akihiro Kishimoto, “A Survey of Parallel A*”

[2] Ariana Weinstock and Rachel Holladay, “Parallel A* Graph Search”