1.BFS

1.             while (true)

2.             {

3.                 Node\* front = frontier.front();

4.                 frontier.erase(frontier.begin());

5.

6.                 explored\_node\_count++;

7.

8.

9.                 auto stop\_time = chrono::high\_resolution\_clock::now();

10.                 auto duration = chrono::duration\_cast<chrono::minutes>(stop\_time - start\_time);

11.                 if (duration.count() >= time\_limit)

12.                 {

13.                     cout << "Time limit reached!" << endl;

14.                     break;

15.                 }

16.

17.                 fill\_children(front);

18. if(check\_optimal\_solution(front))

19. break;

20.

21.                 for (Node\* child : front->children)

22.                 {

23.                     frontier.push\_back(child);

24.                 }

25.             max\_frontier\_size = frontier.size() > max\_frontier\_size ? frontier.size() : max\_frontier\_size;

26

27.             }

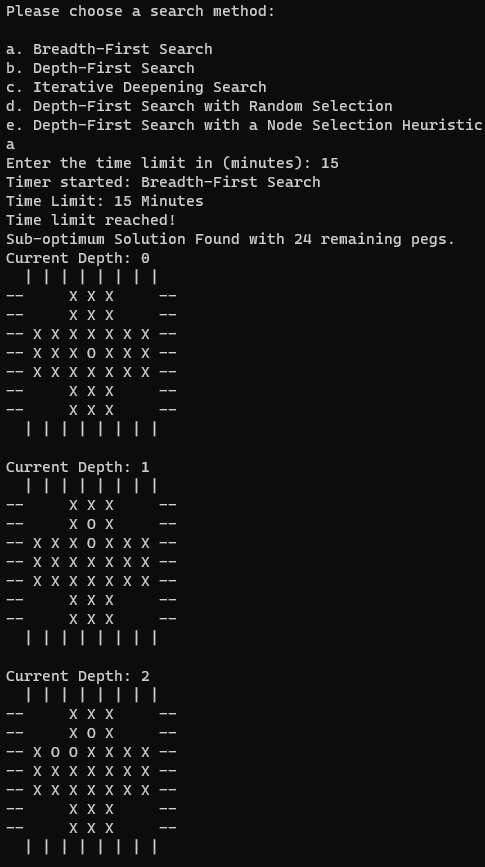
28.

This code is our BFS implementation. It uses a frontier list in a while loop.

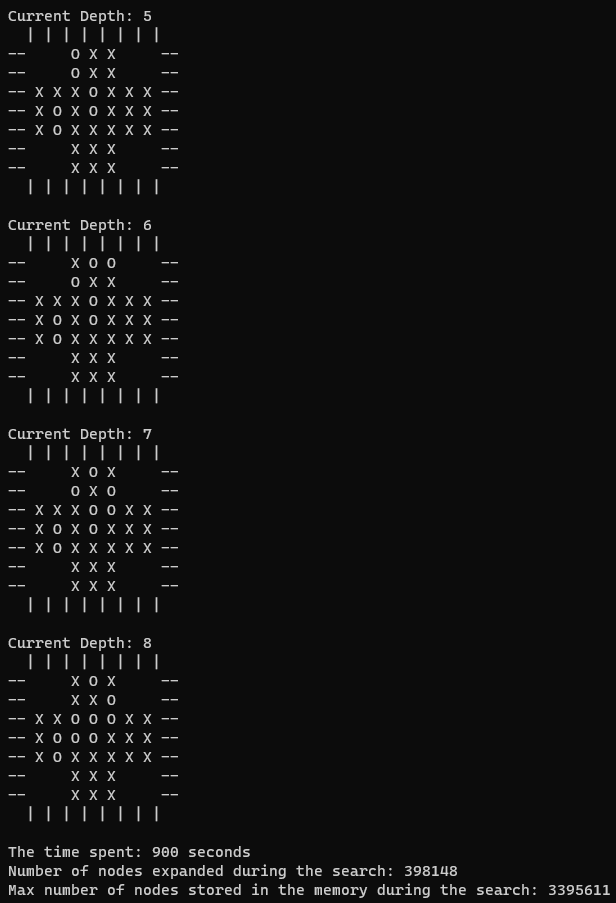
* First, we pop the front node from the frontier vector.
* Since we explored this node, we increment our counter.
* Then we check whether the program reached its time limit.
* Then fill children function finds the possible moves and adds them as children.
* We control if front is optimal solution or not.
* Check optimal solution function also checks for the sub-optimality.
* Then we push those children to the back of the vector. (Since it is a BFS algorithm)
* We check whether the frontier size is maximum in the current run.

**We coded this AI in C++ and within our tests we couldn’t get to 60 minutes runtime it was always out of memory.**

**So we tested in different time limits.**







2. DFS

1. while (true)

2. {

3.

4. Node\* top = frontier.front();

5. frontier.erase(frontier.begin());

6.

7. explored\_node\_count++;

8.

9.

10. auto stop\_time = chrono::high\_resolution\_clock::now();

11. auto duration = chrono::duration\_cast<chrono::minutes>(stop\_time - start\_time);

12. if (duration.count() >= time\_limit)

13. {

14. cout << "Time limit reached!" << endl;

15.

16. break;

17. }

18.

19. fill\_children(top);

20.

21. if(check\_optimal\_solution(top))

22. break;

23.

24. for (int i = top->children.size() - 1; i >= 0 ; i--)

25. {

26. frontier.emplace(frontier.begin(), top->children.at(i));

27. }

28.

29. max\_frontier\_size = frontier.size() > max\_frontier\_size ? frontier.size() : max\_frontier\_size;

30. }

31.

This code is our DFS implementation. It uses a frontier list in a while loop.

• First, we pop the front node from the frontier vector.

• Since we explored this node, we increment our counter.

• Then we check whether the program reached its time limit.

• Then fill children function finds the possible moves and adds them as children.

• We control if front is optimal solution or not. Check optimal solution function also checks for the sub-optimality.

• Then we push those children to the front of the vector. (Since it is a DFS algorithm)

• We check whether the frontier size is maximum in the current run.

3. IDS

int depth\_limit = 0;

bool cutoff = false;

while (true)

{

while (true)

{

Node\* top = frontier.front();

frontier.erase(frontier.begin());

explored\_node\_count++;

auto stop\_time = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::minutes>(stop\_time - start\_time);

if (duration.count() >= time\_limit)

{

cout << "Time limit reached!" << endl;

break;

}

if (depth\_limit > top->depth)

{

fill\_children(top);

if (check\_optimal\_solution(top))

{

cutoff = true;

break;

}

for (int i = top->children.size() - 1; i >= 0; i--)

{

frontier.emplace(frontier.begin(), top->children.at(i));

}

}

max\_frontier\_size = frontier.size() > max\_frontier\_size ? frontier.size() : max\_frontier\_size;

if (frontier.empty())

break;

}

depth\_limit++;

frontier.push\_back(initial);

if (cutoff)

{

break;

}

}

break;

This code is our Iterative Deepening Search implementation. It uses a frontier list in a while loop.

* First, we initialize the depth limit and cutoff values.
* Then for every depth:
  + First, we pop the front node from the frontier vector.
  + Since we explored this node, we increment our counter.
  + Then we check whether the program reached its time limit.
  + Then fill children function finds the possible moves and adds them as children.
  + We control if front is optimal solution or not. If it is set cutoff as true.
  + Check optimal solution function also checks for the sub-optimality.
  + Then we push those children to the front of the vector. (Since it’s like a DFS algorithm)
  + We check whether the frontier size is maximum in the current run.
  + If it is that means it can to another depth, so we break that inner loop.
* Increment our limit
* Start the frontier again with the initial state.
* If cutoff, if optimal solution is found, break the loop.

4. DFS w/Random

1. while (true)

2. {

3. Node\* top = frontier.front();

4. frontier.erase(frontier.begin());

5.

6. explored\_node\_count++;

7.

8.

9. auto stop\_time = chrono::high\_resolution\_clock::now();

10. auto duration = chrono::duration\_cast<chrono::minutes>(stop\_time - start\_time);

11. if (duration.count() >= time\_limit)

12. {

13. cout << "Time limit reached!" << endl;

14.

15. break;

16. }

17.

18. fill\_children(top);

19.

20. if (check\_optimal\_solution(top))

21. break;

22.

23. vector<Node\*> shuffled\_children = top->children;

24.

25. unsigned seed = chrono::system\_clock::now().time\_since\_epoch().count();

26. shuffle(shuffled\_children.begin(), shuffled\_children.end(), default\_random\_engine(seed));

27.

28. for (Node\* child : shuffled\_children)

29. {

30. frontier.emplace(frontier.begin(), child);

31. }

32.

33. max\_frontier\_size = frontier.size() > max\_frontier\_size ? frontier.size() : max\_frontier\_size;

34. }

35. break;

36.

This code is our DFS with random selection implementation. It uses a frontier list in a while loop as always.

* First, we pop the front node from the frontier vector.
* Since we explored this node, we increment our counter.
* Then we check whether the program reached its time limit.
* Then fill children function finds the possible moves and adds them as children.
* We control if front is optimal solution or not. Check optimal solution function also checks for the sub-optimality.
* We shuffle the children vector in the frontier and create a new vector.
* Then we push those children to the front of the vector. (Since it is a DFS algorithm)
* We check whether the frontier size is maximum in the current run.

5. DFS w/Heuristic

1. while (true)

2. {

3. Node\* top = frontier.front();

4. frontier.erase(frontier.begin());

5.

6. explored\_node\_count++;

7.

8.

9. auto stop\_time = chrono::high\_resolution\_clock::now();

10. auto duration = chrono::duration\_cast<chrono::minutes>(stop\_time - start\_time);

11. if (duration.count() >= time\_limit)

12. {

13. cout << "Time limit reached!" << endl;

14. break;

15. }

16.

17. fill\_children(top);

18.

19. if (check\_optimal\_solution(top))

20. break;

21.

22. for (int i = 0; i < top->children.size(); i++)

23. {

24. top->children.at(i)->set\_heuristic\_point();

25. }

26. vector<Node\*> sorted\_children = top->children;

27. sort(sorted\_children.begin(), sorted\_children.end(), compare);

28.

29. for (int i = sorted\_children.size() - 1; i >= 0; i--)

30. {

31. frontier.emplace(frontier.begin(), sorted\_children.at(i));

32. }

33.

34. max\_frontier\_size = frontier.size() > max\_frontier\_size ? frontier.size() : max\_frontier\_size;

35. }

36. break;

37.

1. void Node::set\_heuristic\_point()

2. {

3. if (moved\_peg\_index % 7 == 0 || moved\_peg\_index % 7 == 6)

4. {

5. heuristic\_point = 0.8;

6. }

7. else if (moved\_peg\_index % 7 == 1 || moved\_peg\_index % 7 == 5)

8. {

9. heuristic\_point = 0.6;

10. }

11. else if (moved\_peg\_index % 7 == 2 || moved\_peg\_index % 7 == 4)

12. {

13. heuristic\_point = 0.4;

14. }

15. else if (moved\_peg\_index % 7 == 3)

16. {

17. heuristic\_point = 0.2;

18. }

19. if (remaining\_pegs < 7)

20. {

21. heuristic\_point = 1 - heuristic\_point;

22. }

23. }

24.

This code is our DFS with heuristics implementation. It uses a frontier list in a while loop as always.

• First, we pop the front node from the frontier vector.

• Since we explored this node, we increment our counter.

• Then we check whether the program reached its time limit.

• Then fill children function finds the possible moves and adds them as children.

• We control if front is optimal solution or not. Check optimal solution function also checks for the sub-optimality.

• Then program sets each node a heuristic point according to the set heuristic point function in the node class. Those children are sorted according to their points.

• Then that sorted vector is pushed to the front of the vector. (Since it is a DFS algorithm)

• We check whether the frontier size is maximum in the current run.