Lab Class

assignment

→ output - 9 cost (swap) হবে steps গুলো দেখাবো

→ input → n, init., goal

- solving 15-puzzle by A^* search (combinatorial problem)

start config. (given)

	2	3	4
1	5	7	8
10	6	11	12
9	13	14	15

- - - - →

goal config.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

4 → tile

(empty cell) → blank tile (0)

Topic

i. modeling AI problems as search

ii. problem relaxation and search heuristics

- it is basically (n^2-1)-puzzle problem where n belongs to N.

⇒ given config. থেকে A* search করে target config. - এ যাবো

⇒ target unreachable হলে সেটা বলবো

A-star Search

- branch-and-bound method-এর example

given example consider করে,

offline-এ input এই format-এ দিবে

s <0 2 3 4 1 5 7 8 10 6 11 12 9 13 14 15>

g <■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0>

- combinatorial problem হিসেবে specific problem এবং specific approach-এ problem টা solve করা যায়

- AI-এ approach ভিন্ন হবে। এক্ষেত্রে "general problem solving"-এ একটা generic approach-এ আসা যাবে।

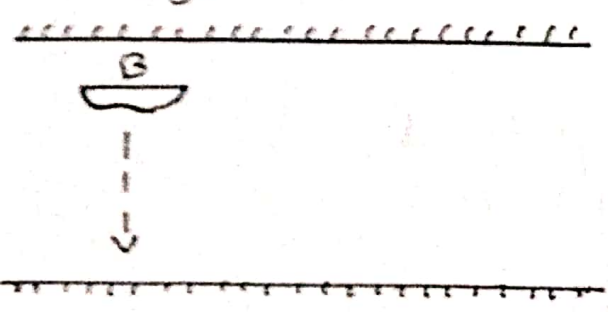
=> এক্ষেত্রে,

we will formulate the problem as a graph and apply a graph search technique. In this case, we will use DFS for our convenience as well as efficiency. (branch-and-bound technique)

Example - 1

wolf-goat-cabbage problem

- related problem: missionaries and cannibal problem



wolf, goat, cabbage-কে নদীর এপাড় থেকে ওপাড়ে চালান করতে হবে।

wolf → goat কে খায়

goat → cabbage কে খায়

at a time, একজনকে (w, g, c)-কে boat carry করতে পারবে। wolf, goat unattended অবস্থায় (তীরে নৌকা না থাকা) থাকতে পারবে না। একই ব্যাপার g, c - এর জন্যও প্রযোজ্য।

=> কীভাবে চালান করা যাবে?

• Problem টা general approach - এ
solve করবো

=> Graph বানাতে state space বানাবো।
তার জন্য all possible states
চিন্তা করতে হবে।

=> এর পর এই state গুলো দিয়ে start থেকে
target state - এ যাবো (DFS চালাবো)

=> invalid state গুলো বাদ দিয়ে state
space ছোট করলে efficiently problem
solve করা যায়।

=> এখানে, Graph/search tree expand
করতে থাকবো

=> তাই শুরুতে কোনো data-structure - এ
রাখবো না

=> problem efficiently solve করার
কথা মাথায় রেখে search করবো

Example-2

Water pouring puzzles

- jug-এর গায়ে per litre দাগ থাকেনা
- তাই fraction amount pass করা যাবে না

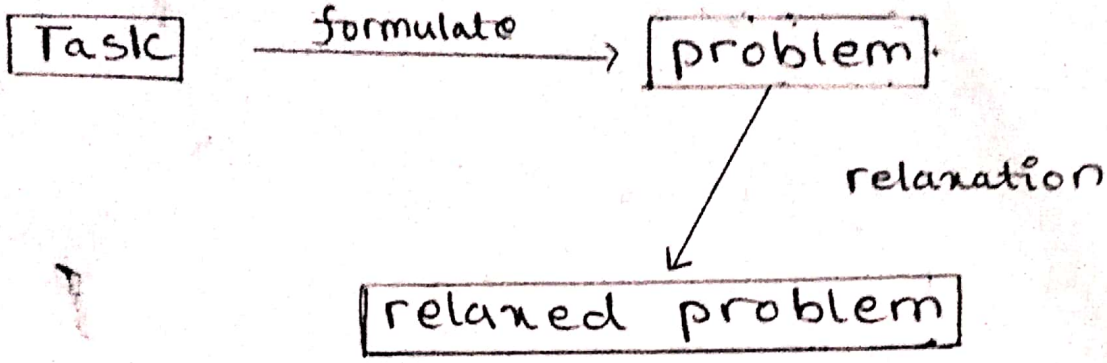
=> যা আছে পুরাটা ঢালা লাগবে

=> যেটায় ঢালছি - তা full হওয়া অবধি ঢালা যাবে

- wikipedia দ্রষ্টব্য
- search algorithm গুলোর properties (BFS, DFS, Dijkstra) দেখবো

- State space অনেক বড় হলে bnb method-এ efficiently problem solve করা যায়। bnb-তে কেবল promising branch-এ যায়। search space-এর যতটুকু expand না করলেই নয়, ততটুকুই expand করব।

- problem relaxation → problem-এর constraint/condition relaxation



- puzzle problem কে relax করে কম সময়ে problem solve করবো
- tile operation/movement ⟶

i. corner হলে 2 দিকে
ii. edge হলে 3 দিকে
iii. otherwise 4 দিকে

⇒ blank tile হলে তো কিছু নাই। কোনো tile কে সরানো মানে কেবল তার পাশে blank থাকে

- relaxed problem solve করার minimum cost কে bound ধরে heuristic হিসাবে problem solve করবো

i. & ii. - ২ ভাবেই করায় system থাকবে

• we have →

i. displacement heuristic

example-টা চিন্তা করি →

tile-1 + tile-2 + ---- + tile-15

1 (displaced) জায়গায় নাই ; 0 (আছে) ; 1

$$= \underset{1}{1} + \underset{2}{0} + \underset{3}{0} + \underset{4}{0} + \underset{5}{1} + \underset{6}{1} + \underset{7}{0} + \underset{8}{0}$$

$$+ 1 + 1 + 0 + 0 + 1 + 1 + 1 + \underset{\text{blank}}{(1)}$$

= 9-সংখ্যক tile জায়গা মতো নাই
তাই at least 9বার tile swap
করা লাগবে (bound)
⇒ minimum 9 movements required
⇒ tile গুলোর movement পরস্পর
independent ধরে এই heuristic পাই।

ii. manhattan heuristic

• i.-এ blank tile adjacency-র condition ছিল। ii.-এ এই constraint তুলে দেওয়া হয় (relaxation)

এক্ষেত্রে → | 4 | 2 | 3 | 1 | হলে (goal)

tile-1 + tile-2 + tile-3 + tile-4 + ---

(1↑ + 3→) ; 0 ; 0 ; (3←)

⇒ এখানেও movement independent.