Knight moves (BFS in 2d Grid) — Knight moves problem in Onlinejudge

Knight এর সংক্ষিপ্ত পথ**:** একটি **BFS** সমাধান

সমস্যাটি কী?

আমাদের এই সমস্যাটির কাজ হলো একটি ক্লাসিক 'shortest path' এর সমাধান করা। সমস্যাটির নির্দিষ্ট লক্ষ্য হলো: দাবাবোর্ডের দুটি স্কয়ার a এবং b দেওয়া থাকলে, a থেকে b তে পৌঁছাতে নাইটের সর্বনিম্ন কতগুলো চাল লাগবে!

এই সমস্যাটিকে একটি **unweighted graph** এর সংক্ষিপ্ততম পথ খোঁজার সমস্যা হিসাবে মডেল করা যেতে পারে। এখানেঃ

- নোড **(Nodes):** দাবাবোর্ডের ৬৪টি স্কয়ার হলো এই গ্রাফের নোড।
- এজ **(Edges):** দুটি স্কয়ারের মধ্যে একটি বৈধ নাইটের চাল হলো একটি এজ।

যেহেতু প্রতিটি চালের cost সমান (অর্থাৎ ১ চাল), তাই এই ধরনের গ্রাফে সংক্ষিপ্ততম পথ খোঁজার জন্য সবচেয়ে আদর্শ অ্যালগরিদম হলো ব্রেডথ-ফার্স্ট সার্চ **(Breadth-First Search** বা **BFS)।**

BFS কী?

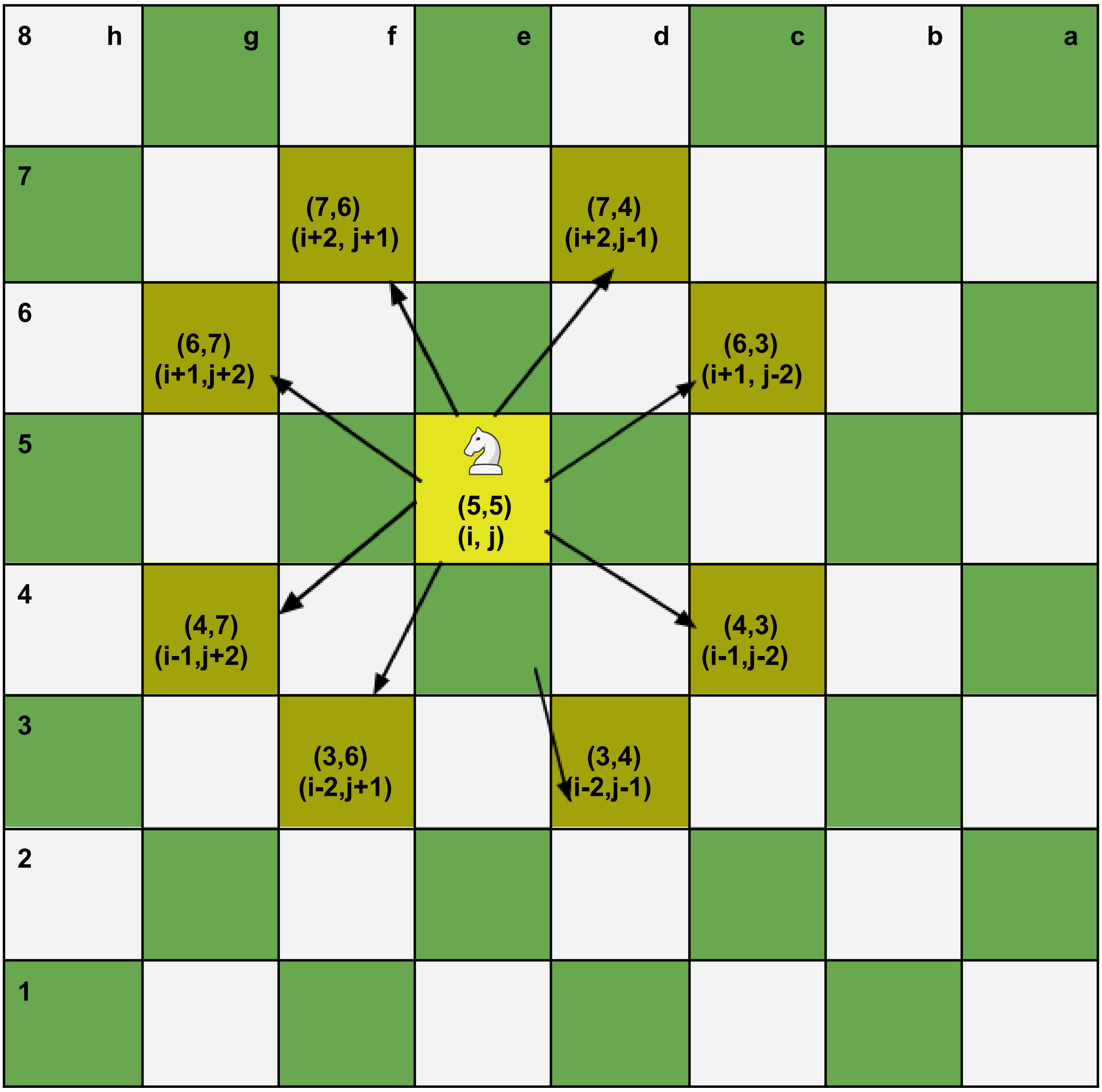
BFS, বা ব্রেডথ-ফার্স্ট সার্চ, হলো একটি গ্রাফ ট্রাভার্সাল অ্যালগরিদম যা শুরু নোড থেকে শুরু করে গ্রাফটিকে দূরত্ব অনুযায়ী স্তর-স্তরে **(level by level)** search করে। এটি কিউ (Queue) ডেটা স্ট্রাকচার ব্যবহার করে এবং নিশ্চিত করে যে সংক্ষিপ্ততম পথে পৌঁছানোর আগে এটি কোনো দীর্ঘ পথে যাবে না। ওজনহীন গ্রাফে (যেমন এই নাইটের সমস্যা), এটিই সংক্ষিপ্ততম পথ খুঁজে বের করার সবচেয়ে কার্যকর উপায়।

বোর্ড এবং নাইটের চাল রিপ্রেজেন্ট করা

BFS অ্যালগরিদম ব্যবহার করার আগে, আমাদের গ্রাফটিকে কোডে রিপ্রেজেন্ট করতে হবে। অর্থাৎ, একটি নাইট কীভাবে এক নোড থেকে অন্য নোডে (এক স্কয়ার থেকে অন্য স্কয়ারে) যেতে পারে, তা সংজ্ঞায়িত করতে হবে।

নাইটের ৮টি সম্ভাব্য চাল

একটি নাইট, দাবাবোর্ডের যেকোনো স্কয়ার `(i, j)` (সারি `i` এবং কলাম `j`) থেকে সর্বোচ্চ ৮টি ভিন্ন স্কয়ারে যেতে পারে। নিচের চিত্রটি এটি পরিষ্কারভাবে তুলে ধরেছে:



`(i, j)` পজিশন থেকে সম্ভাব্য ৮টি চাল হলো:

- `(i-2, j+1)`
- `(i-2, j-1)`
- `(i-1, j+2)`
- `(i-1, j-2)`
- `(i+1, j+2)`
- `(i+1, j-2)`
- `(i+2, j+1)`
- `(i+2, j-1)`

চিত্র থেকে কোডে রূপান্তর

এই ৮টি "ডেল্টা" মুভকে (অর্থাৎ সারি ও কলামের পরিবর্তন) C++ কোডে দুটি ধ্রুবক (constant) অ্যারের মাধ্যমে খুব সহজে রিপ্রেজেন্ট করা যায়। `kr[]` অ্যারেটি সারির পরিবর্তন এবং `kc[]` অ্যারেটি কলামের পরিবর্তন সংরক্ষণ করবে।

```
// kr[] অ্যারে ৮টি সম্ভাব্য সারির পরিবর্তন সংরক্ষণ করে
const int kr[] = {2, 2, -2, -2, 1, 1, -1, -1};

// kc[] অ্যারে ৮টি সম্ভাব্য কলামের পরিবর্তন সংরক্ষণ করে
const int kc[] = {1, -1, 1, -1, 2, -2, 2, -2};
```

এখন, `i = 0` থেকে `7` পর্যন্ত একটি লুপ চালিয়ে `new_r = r + kr[i]` এবং `new_c = c + kc[i]` করলেই আমরা যেকোনো স্কয়ার `(r, c)` থেকে ৮টি সম্ভাব্য নতুন স্কয়ার পেয়ে যাব।

Breadth-First Search (BFS) এর ব্যবহার

BFS এই সমস্যার জন্য উপযুক্ত কারণ এটি গ্রাফটিকে layer অনুযায়ী explore করে।

- **Layer 0:** আমাদের শুরুর স্কয়ার (দূরত্ব ০)।
- **Layer 1:** শুরুর স্কয়ার থেকে ১ চালে পৌঁছানো যায় এমন সব স্কয়ার (দূরত্ব ১)।
- **Layer 2:** শুরুর স্কয়ার থেকে ২ চালে পৌঁছানো যায় এমন সব স্কয়ার (দূরত্ব ২)।
- ... এবং এভাবে চলতে থাকে।

BFS এর এই বৈশিষ্ট্যের কারণে, আমরা যখনই প্রথমবার আমাদের গন্তব্যের স্কয়ারে পৌঁছাব, আমরা নিশ্চিত থাকতে পারি যে এটিই সংক্ষিপ্ততম পথ।

আমাদের সার্চ চালানোর জন্য, কোডে তিনটি প্রধান জিনিস ব্যবহার করতে হবে:

১. **`int dis[n][n]`:** একটি 2D অ্যারে, যা শুরুর স্কয়ার থেকে অন্য প্রতিটি স্কয়ারের দূরত্ব (ন্যূনতম চালের সংখ্যা) সংরক্ষণ করে। শুরুতে আমরা সব দূরত্ব `INT_MAX` (অসীম) ধরে নিই।

২. **`int color[n][n]`:** একটি 2D অ্যারে, যা প্রতিটি স্কয়ারের অবস্থা (state) ট্র্যাক করে। এটি BFS-এর জন্য অপরিহার্য: `-1` (সাদা): স্কয়ারটি এখনো ভিজিট করা হয়নি। `1` (ধূসর): স্কয়ারটি ভিজিট করা হয়েছে এবং এটি

বর্তমানে queue-তে আছে (এর neighbour খোঁজা বাকি)। 2 (কালো): স্কয়ারটি প্রসেস করা শেষ (এর সব neighbour queue-তে যোগ করা শেষ)।

৩. **`queue<pair<int, int>> q`**: একটি কিউ, যা "ধূসর" স্কয়ারগুলো সংরক্ষণ করে। অর্থাৎ, যে স্কয়ারগুলো আমাদের পরবর্তী ধাপে ভিজিট করতে হবে।

PseudoCode

এই PseudoCode-টি BFS অ্যালগরিদম ব্যবহার করে একটি নাইটকে সোর্স স্কয়ার থেকে ডেসটিনেশন স্কয়ারে নিয়ে যাওয়ার জন্য সর্বনিম্ন চালের সংখ্যা নির্ণয় করে।

```
FUNCTION bfs_knight(startR, startC, endR, endC):
    N = 8
    DEFINE dist[N][N] AS ARRAY
    DEFINE color[N][N] AS ARRAY
    DEFINE Queue Q

    FOR r FROM 0 TO N-1:
        FOR c FROM 0 TO N-1:
            dist[r][c] = INFINITY
            color[r][c] = -1

    dist[startR][startC] = 0
    color[startR][startC] = 1
    ENQUEUE (startR, startC) INTO Q

    WHILE Q IS NOT EMPTY:
        (r, c) = DEQUEUE FROM Q

        IF (r, c) IS EQUAL TO (endR, endC):
            RETURN dist[r][c]

        FOR i FROM 0 TO 7:
            newR = r + kr[i]          // kr[] = {2, 2, -2, -2, 1, 1, -1, -1}
            newC = c + kc[i]          // kc[] = {1, -1, 1, -1, 2, -2, 2, -2}

            IF newR, newC IS ON BOARD AND color[newR][newC] IS -1:
                color[newR][newC] = 1
                dist[newR][newC] = dist[r][c] + 1
                ENQUEUE (newR, newC) INTO Q
        color[r][c] = 2
    RETURN -1
```

ধাপে ধাপে বিশ্লেষণ

আমরা একটি উদাহরণ দিয়ে দেখব কীভাবে এই BFS অ্যালগরিদম কাজ করে।

উদাহরণ: সোর্স a1 থেকে ডেসটিনেশন h8 (a1→h8)

- কো-অর্ডিনেট রূপান্তর:
    - a1 ⇒ (startR=0, startC=0) (বোর্ডের বাম-নিচের কোণ)
    - h8 ⇒ (endR=7, endC=7)   (বোর্ডের ডান-উপরের কোণ)

ধাপ ১: ইনিশিয়ালাইজেশন

- dist অ্যারেতে সব মান   সেট করা হয়।
- color অ্যারেতে সব মান -1 (Unvisited) সেট করা হয়।
- **a1(0, 0):** dist[0][0] = 0, color[0][0] = 1।
- Queue Q তে (0, 0) যোগ করা হয়। ⇒ Q: [(0, 0)]

ধাপ ২: দূরত্ব **d=0** এর স্কয়ার প্রসেস করা

বর্তমান স্কয়ার	চালের সংখ্যা **(d)**	প্রক্রিয়া
a1 (0, 0)	0	Q থেকে (0, 0) বের করা হয়।
-	1	a1 থেকে সম্ভাব্য ৮টি চাল গণনা করা হয়: b3(2, 1) এবং c2(1, 2) (বোর্ডের মধ্যে)।
-	1	b3(2, 1) কে ভিজিট করে dist[2][1]=1 সেট করা হয়। c2 (1, 2) কে ভিজিট করে dist[1][2]=1 সেট করা হয়।
-	1	b3 এবং c2 queue-তে যোগ করা হয়।
-	2	(0, 0) এর color **2** (Processed) সেট করা হয়।

⇒ Q: [(2, 1), (1, 2)]

ধাপ ৩: দূরত্ব **$d=1$** এর স্কয়ার প্রসেস করা

বর্তমান স্কয়ার	চালের সংখ্যা **(d)**	প্রক্রিয়া
b3 (2, 1)	1	**Q** থেকে **(2, 1)** বের করা হয়। এর ৮টি সম্ভাব্য চালের মধ্যে a5 (4, 0), c5 (4, 2), d4 (3, 3) ইত্যাদি ভিজিট করা হয়।
-	2	এই নতুন স্কয়ারগুলোর `dist` **2** সেট করা হয় এবং কিউতে যোগ করা হয়।
c2(1,2)	1	**Q** থেকে **(1, 2)** বের করা হয়। এর ৮টি সম্ভাব্য চাল গণনা করা হয়।
-	2	এদের মধ্যে কিছু স্কয়ার আগেই d=2 এ ভিজিট হয়ে থাকতে পারে (যেমন: d4)। কিন্তু যেহেতু color ≠ -1, সেহেতু এটি পুনরায় ভিজিট হবে না। নতুন স্কয়ারগুলোর dist **2** সেট করা হয় এবং কিউতে যোগ করা হয়।

⇒ Q তে এখন দূরত্ব **d=2** এর সব স্কয়ার আছে।

ধাপ ৪: **Continue….**

এই প্রক্রিয়া চলতে থাকবে, প্রতিটি ধাপে d এর মান ১ করে বাড়বে:

- **d=2** এর সব স্কয়ার প্রসেস হবে। তাদের প্রতিবেশী স্কয়ার, যা $d=3$ হবে, তা কিউতে যোগ করা হবে।
- **d=3** এর সব স্কয়ার প্রসেস হবে।
- ... চলতে থাকবে।

যে মুহূর্তে (7, 7) স্কয়ারটি তার কোনো প্রতিবেশীর মাধ্যমে প্রথমবার খুঁজে পাওয়া যাবে, সেই মুহূর্তে:

1. `newR = 7`, `newC = 7` হবে।
2. `color[7][7]` হবে -1 (কারণ এটি প্রথমবার ভিজিট হচ্ছে)।
3. `dist[7][7]` সেট করা হবে dist[r][c] + 1, যেখানে r, c হলো সেই স্কয়ার যা h8 এর ঠিক আগের চালটি।
4. পরবর্তীতে, যখন h8 (7, 7) স্কয়ারটি কিউ থেকে বের করা হবে, তখন `if (r == endR && c == endC)` শর্তটি সত্য হবে।
5. ফাংশনটি dist[7][7] এর মান রিটার্ন করবে।

ফলাফল**:** a1 থেকে h8 পর্যন্ত সর্বনিম্ন চালের সংখ্যা হবে **6।** যেহেতু BFS স্তর অনুসারে চলে, তাই এটি নিশ্চিতভাবেই সর্বনিম্ন সংখ্যা।