|  |
| --- |
| Algorithm: Bidirectional Search |
| Input: Start node *S*, Goal node *G*, Grid map *M* |
| Output: Optimal path from *S* to *G* |
|  |
| 1. Initialize: |
| * Open list (forward) ← {*S*} |
| * Open list (backward) ← {*G*} |
| * Closed list (forward) ← ∅ |
| * Closed list (backward) ← ∅ |
| * *g*(*S*) ← 0, *g*(*G*) ← 0 |
| * *f* (*S*) ← *h*(*S*, *G*), *f* (*G*) ← *h*(*G*, *S*) |
| 1. While both Open lists are not empty: |
| 1. Forward Search: |
| 1. Select node *nf* from Open list (forward) with the smallest *f* (*nf* ). |
| 1. Remove *nf* from Open list (forward) and add to Closed list (forward). |
| 1. For each neighbor *m* of *nf* : |
| 1. If *m* ∈ Closed list (backward): |
| 1. Path found! Reconstruct path from *S* to *G*. |
| 1. Tentative cost *gt* ← *g*(*nf* ) + *cost*(*nf* , *m*). |
| 1. If *m* ∈/ Open list (forward) or *gt* < *g*(*m*): |
| 1. *g*(*m*) ← *gt*. |
| 1. *f* (*m*) ← *g*(*m*) + *h*(*m*, *G*). |
| 1. Parent of *m* ← *nf* . |
| 1. Add *m* to Open list (forward). |
| 1. Backward Search: |
| 1. (Repeat steps 4–14 for the backward direction, treating *G* as start and *S* as goal.) |
| 1. If Open list (forward) or Open list (backward) is empty: |
| 1. Path not found. |
| 1. Output: |
| 1. Return the optimal path from *S* to *G*. |

# Penjelasan Langkah Utama

* 1. Inisialisasi:

 Dua jalur (forward dan backward) dimulai dari *S* dan *G*.

* 1. Pencarian Simultan:

 Jalur maju (forward) mencari dari *S* ke *G*, sedangkan jalur mundur (backward) mencari dari *G* ke *S*.

* 1. Penggabungan Jalur:

 Jika kedua jalur bertemu pada node yang sama, jalur optimal dapat direkonstruksi.

* 1. Efisiensi:

 Pencarian dilakukan pada kedua arah untuk mengurangi waktu pencarian secara keseluruhan.