評価値(評価コスト$\hat{f}(x) = \hat{g}(x) + \hat{h}(x)$)を用いた探索

準備

- […] はノード(状態)のリストを表す.
- 変数 OL, CL, V はノード(状態)のリスト，変数 X は 1 つのノード(状態)を表す.
- s は初期状態を表す.
- +はリスト連結の演算子とする.
- pop(l)は，リスト l の先頭(左端)から要素を一つとりだす関数とする.
- next(x) は，x の隣接ノード(次の状態)のリストを返す関数とする．その際，各ノードに x へのリンクを持たせる．(探索木や経路を生成するのに必要)
- sort(l)は，リスト l の要素を，その評価コスト($\hat{f}(x) = \hat{g}(x) + \hat{h}(x)$)で昇順に並べ替えたリストを返す関数とする.
- uniq(l)は，リスト l の重複を(最左の要素のみを残すことによって)取り除いたリストを返す関数とする.

探索手続き(A*アルゴリズムなど)

```
1   OL ← [s]; CL ← [];
2   if (OL が空) 終了(失敗);
3   X ← pop(OL);
4   if (X が目標状態) 終了(成功);
5   CL ← CL + [X];
6   V ← next(X);
7   V の要素の各々(v)について，以下を行う.
    7.1   if (v が CL に含まれ，かつ，V 中の v の f̂(v)が CL 中の v の f̂(v)以上である)
                V から v を取り除く;          // 探索済みの状態を除外
    7.2   if (v が CL に含まれ，かつ，V 中の v の f̂(v)が CL 中の v の f̂(v)より小さい)
                CL から v を取り除く;         // s～v のコスト ĝ(v)が更新された場合,
                                              // v を CL から OL に戻す
8   OL ← sort(OL + V);        // 評価値(評価コスト)の昇順に並べ替える
9   OL ← uniq(OL);            // 重複は，評価値(評価コスト)の悪いものを除去
10  goto 2
```

※ V 中の v の $\hat{f}(v)$が CL 中の v の $\hat{f}(v)$より小さい場合、$\hat{h}(v)$は変化しないので、$\hat{g}(v)$が更新されたことになる.

※ $\hat{g}(v)$が更新された場合，先に v を経由して探索された状態を再び探索し直す必要が生じるため，7.2 のように，v を再び OL に戻す必要がある.

分枝限定探索$(\hat{f}(x) = \hat{g}(x)\ (\hat{h}(x) \equiv 0))$

1 OL ← [s]; CL ← [];
2 if (OL が空) 終了(失敗);
3 X ← pop(OL);
4 if (X が目標状態) 終了(成功);
5 CL ← CL + [X];
6 V ← next(X);
7 V の要素の各々(v)について，以下を行う．
7.1 if (v が CL に含まれる) V から v を取り除く; // 探索済みの状態を除外
8 OL ← sort(OL + V); // 評価値(評価コスト)の昇順に並べ替える
9 OL ← uniq(OL); // 重複は，評価値(評価コスト)の悪いものを除去
10 goto 2

※ 7.1 で v が CL に含まれている場合，V 中の v の$f(v)$ $(= g(v))$は常に CL 中の v の $f(v)$ $(= g(v))$以上なので，上の 7.2 のように，v を CL から OL に戻す必要はない．

※ 8 の並べ替えを除けば，グラフの(評価コストを用いない)探索に類似している．