木・グラフの探索

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)
- 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.

木の縦型探索

```
1  OL ← [s];                  // OL は次に探索する状態の候補
2  if (OL が空) 終了(失敗);
3  X ← pop(OL);               // X は探索対象
4  if (X が目標状態) 終了(成功);
5  V ← next(X);               // V は X の次の状態のリスト
6  OL ← V + OL;               // 前に追加(LIFO)
7  goto 2.
```

木の横型探索

```
1  OL ← [s];                  // OL は次に探索する状態の候補
2  if (OL が空) 終了(失敗);
3  X ← pop(OL);               // X は探索対象
4  if (X が目標状態) 終了(成功);
5  V ← next(X);               // V は X の次の状態のリスト
6  OL ← OL + V;               // 後に追加(FIFO)
7  goto 2.
```

グラフの縦型探索

```
1    OL ← [s]; CL ← [];                    // CL は探索済みの状態
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];                        // X を探索済みとする
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 ← V + OL;
9    OL ← uniq(OL);                        // 重複は後から追加された方を残して除去
10   goto 2.
```

※ 7 以降は以下でもよい．

```
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;
       7.2    if (v が OL に含まれる) OL から v を取り除く;    // 重複をあらかじめ除去
8    OL ← V + OL;
9    goto 2
```

グラフの横型探索

```
1    OL ← [s]; CL ← [];                    // CL は探索済みの状態
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];                        // X を探索済みとする
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 ← OL + V;
9    OL ← uniq(OL);                        // 重複は後から追加された方を除去
10   goto 2
```

※ 7 以降は以下でもよい．

```
7    V の要素の各々(v)について，以下を行う．
       7.1    if (v が CL または OL に含まれる) V から v を取り除く;
8    OL ← OL + V;
9    goto 2
```