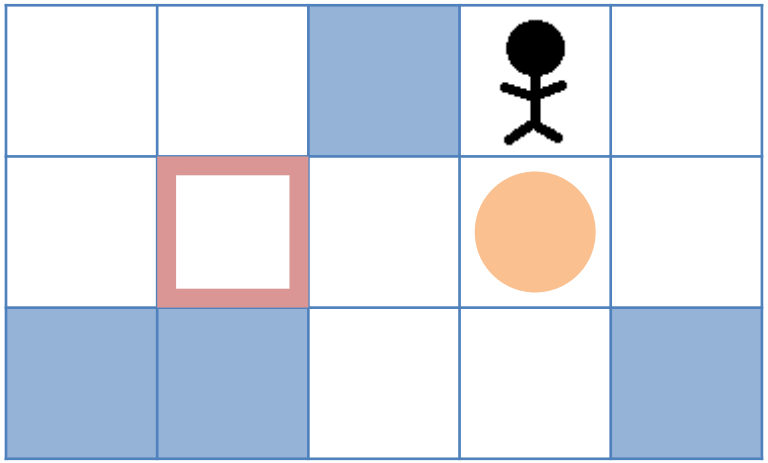
倉庫番 (Sokoban)

JOI 春合宿 2012 Day 3

解説： 保坂 和宏

問題概要

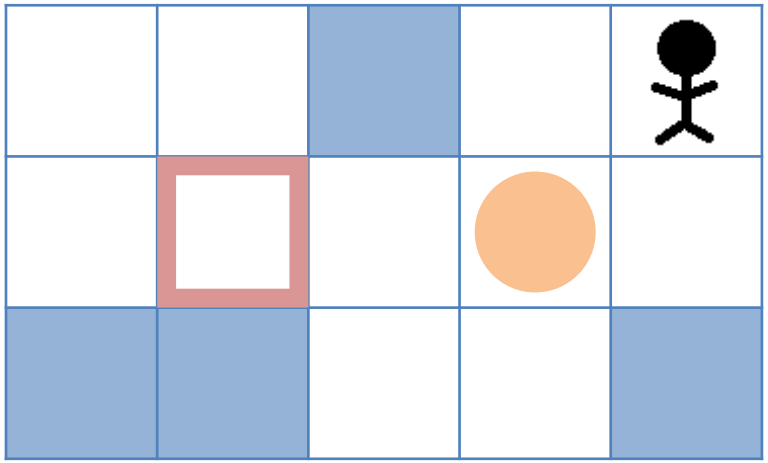
- 倉庫番



  - 箱を押して目標地点に動かす

問題概要

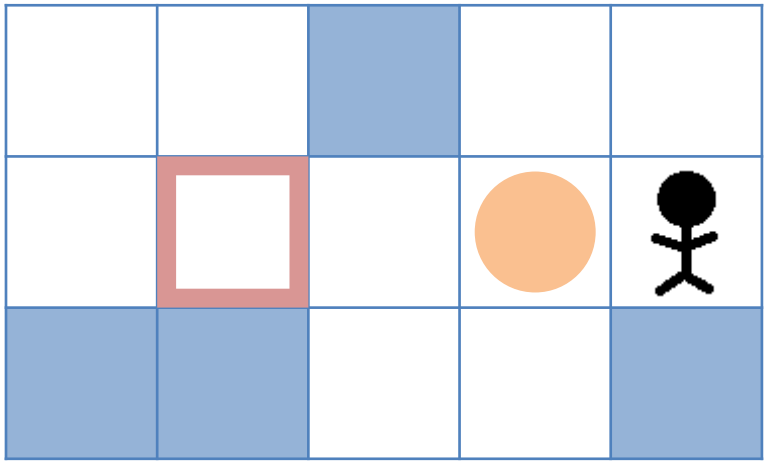
- 倉庫番



  - 箱を押して目標地点に動かす

問題概要

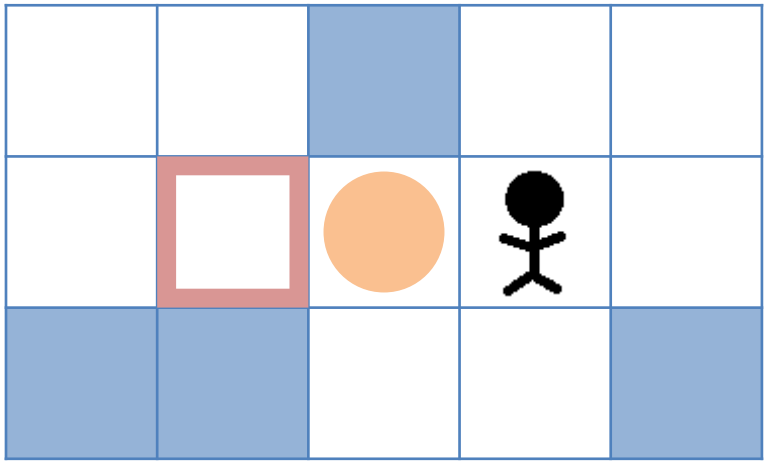
- 倉庫番



  - 箱を押して目標地点に動かす

問題概要

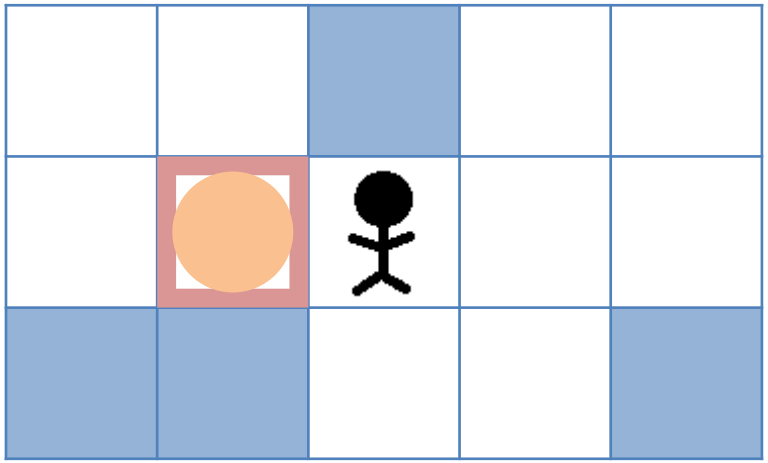
- 倉庫番



  - 箱を押して目標地点に動かす

問題概要

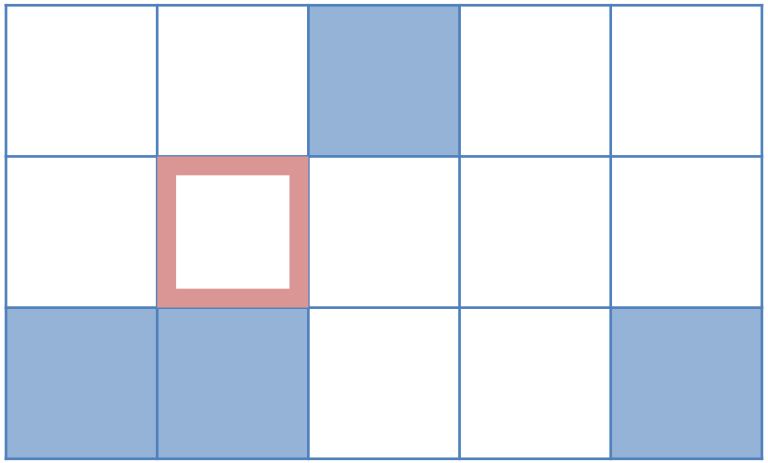
- 倉庫番



  - 箱を押して目標地点に動かす

問題概要

- 倉庫番の問題が何通りあるか数える



  - 壁と目標地点が与えられ，人と箱を置く
  - 盤面のサイズ： $M, N \leqq 1{,}000$

単純な解法

1. 人と箱を置く場所をすべて試す
2. 実際に倉庫番の問題を解く

- 問題は $O(M^2N^2)$ 通り
- 解くのは，人と箱の位置を状態として探索すれば $O(M^2N^2)$
- 全体で $O(M^4N^4)$
  - 0 点 ～ 10 点？

単純な解法

- 単純な解法のどこが無駄か？

  →同じ状態に対して何回も計算

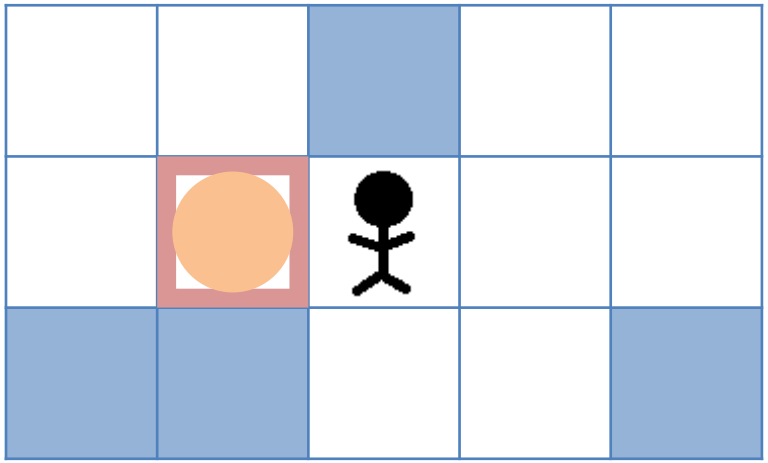
- 解けるような人と箱の位置の組を効率的に列挙したい

部分点解法

- ある人と箱の位置から解けるということは，目標地点から箱を引っぱってきてその人と箱の位置を作れることと同じ

部分点解法

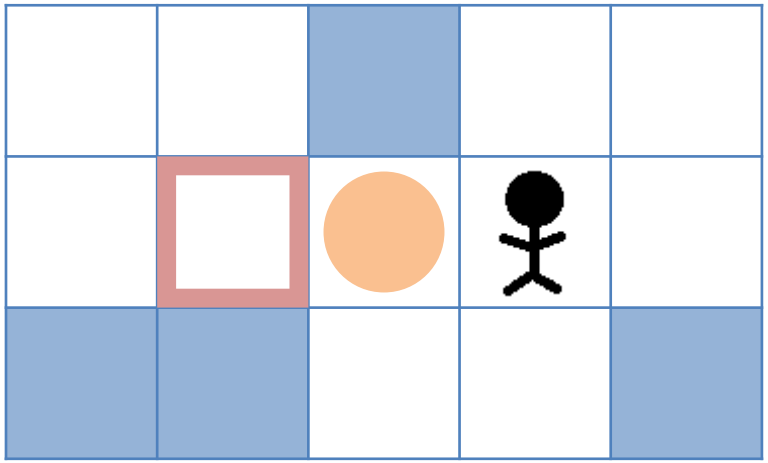
- 倉庫番の逆



  - 目標地点から箱を引っぱる

部分点解法

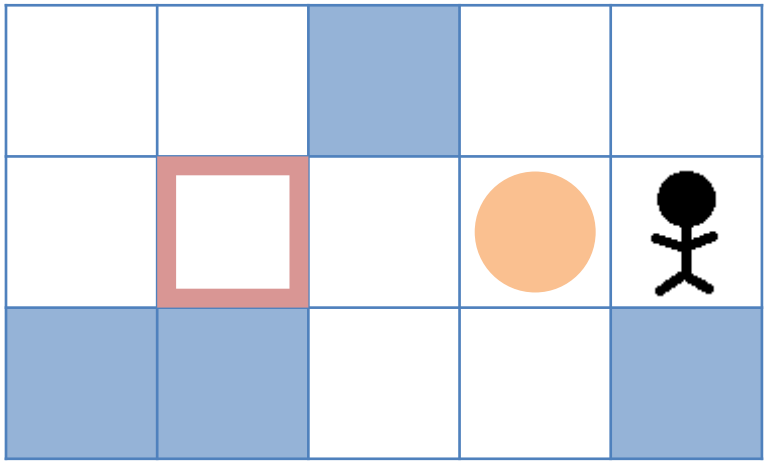
- 倉庫番の逆



  - 目標地点から箱を引っぱる

部分点解法

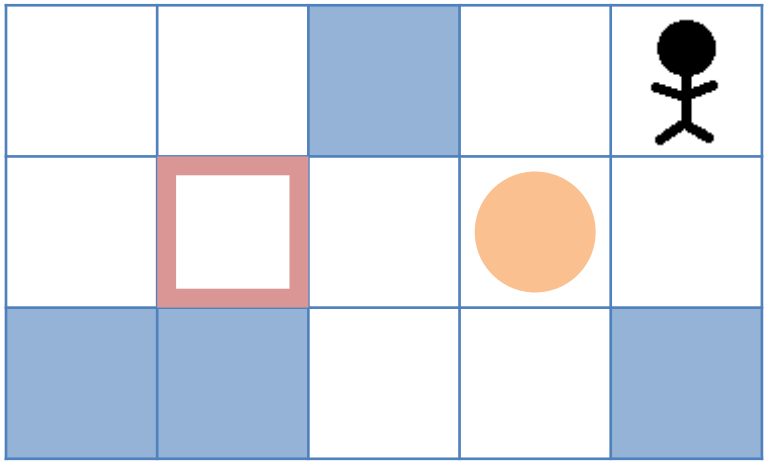
- 倉庫番の逆



  - 目標地点から箱を引っぱる

部分点解法

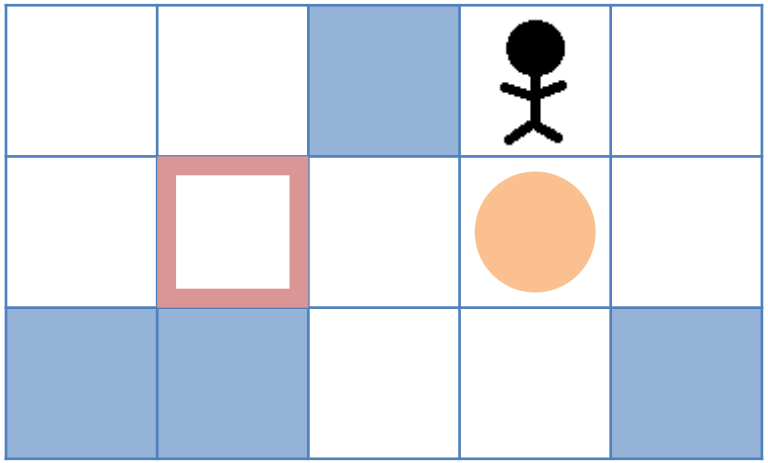
- 倉庫番の逆



  - 目標地点から箱を引っぱる

部分点解法

- 倉庫番の逆



  - 目標地点から箱を引っぱる

部分点解法

- 倉庫番の逆
  - 初期状態：箱は目標地点，人は箱に隣接
  - 移動：
    - 人が動く
    - 人が箱を引っぱって動く

- 幅優先探索などで $O(M^2N^2)$
  - 10 点 ～ 20 点？

アイデア

- 倉庫番の逆
  - 初期状態：箱は目標地点，人は箱に隣接
  - 移動：
    - 人が動く
    - 人が箱を引っぱって動く
- 人は壁と箱で区切られた連結成分内を自由に動ける

アイデア

- 状態
  - 箱のあるマスと人のいるマス
    - 各状態に対して答に 1 を足す

↓

  - 箱のあるマスと人のいる連結成分
    - 各状態に対して答に連結成分のサイズを足す

アイデア

- 目標地点と繋がっていないマスは無視
  - 壁にしてしまえばよい
  - 全体が連結になる

- 連結な状態から箱を置いて 1 マスを塞いでも連結成分は高々 4 個
  - 状態数が $O(MN)$

アイデア

- グリッドグラフを考える
  - 次数が高々 4
  - 箱を 1 個置く＝頂点を 1 個取り除く
  - 頂点を 1 個取り除いたときの状況を知りたい
    - 箱の 4 方向のマスの接続関係と，それぞれの連結成分のサイズ
    - 関節点 (取り除いたら非連結) に関連している？！

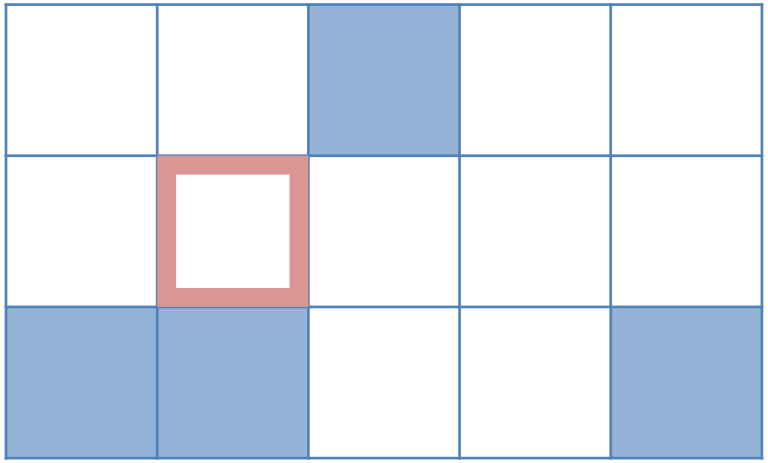
接続関係

- 箱の周囲 4 マスの接続関係
  - グラフの各頂点 u に対してそれを取り除いたときの隣接頂点たちの接続関係

- より一般に「頂点 u を取り除いたときに頂点 a と b は同じ連結成分に属するか」というクエリに答える方法を紹介
  - 今回はこれを行わずとも解けるようです

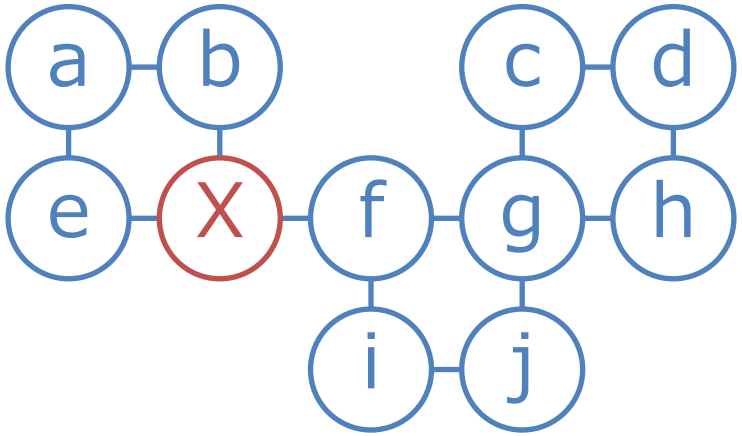
DFS 木

- DFS 木を作る
  - 根はどこでもよいので目標地点からでも

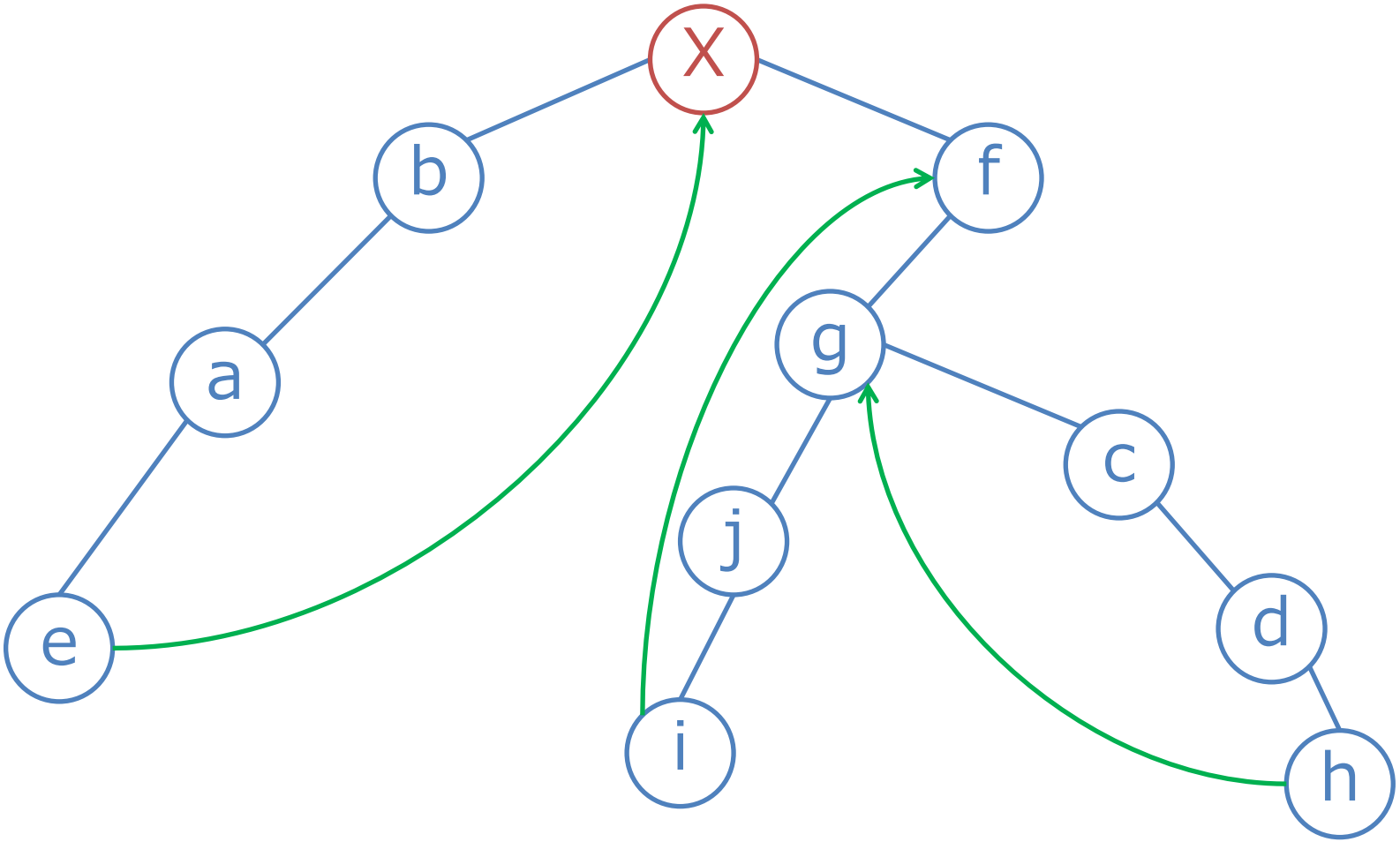


DFS 木

- DFS 木を作る
  - 根はどこでもよいので目標地点からでも



DFS 木



DFS 木

- DFS 木による 2 種類の辺
  - 木の辺
  - 後退辺
    - 祖先へ向かう

- 頂点を訪れた順に番号を付ける
- Lowlink を求める

DFS 木

- *dis*[*u*] ：頂点 *u* を訪れた時刻 (discover)
- *fin*[*u*] ：頂点 *u* を去った時刻 (finish)

- *low*[*u*] ：頂点 *u* の "Lowlink"
  - 頂点 *u* から「木の辺は子孫方向に何度でも」「後退辺は 1 回だけ」用いて到達できる頂点のうちの *dis* の最小値

DFS 木

function *DFS*(*u*)

dis[*u*] := *k*,　　*k* := *k* + 1.

low[*u*] := *dis*[*u*].

for each *v* : (*u* に隣接している頂点):

if (頂点 *v* をまだ訪れていない):

DFS(*v*).

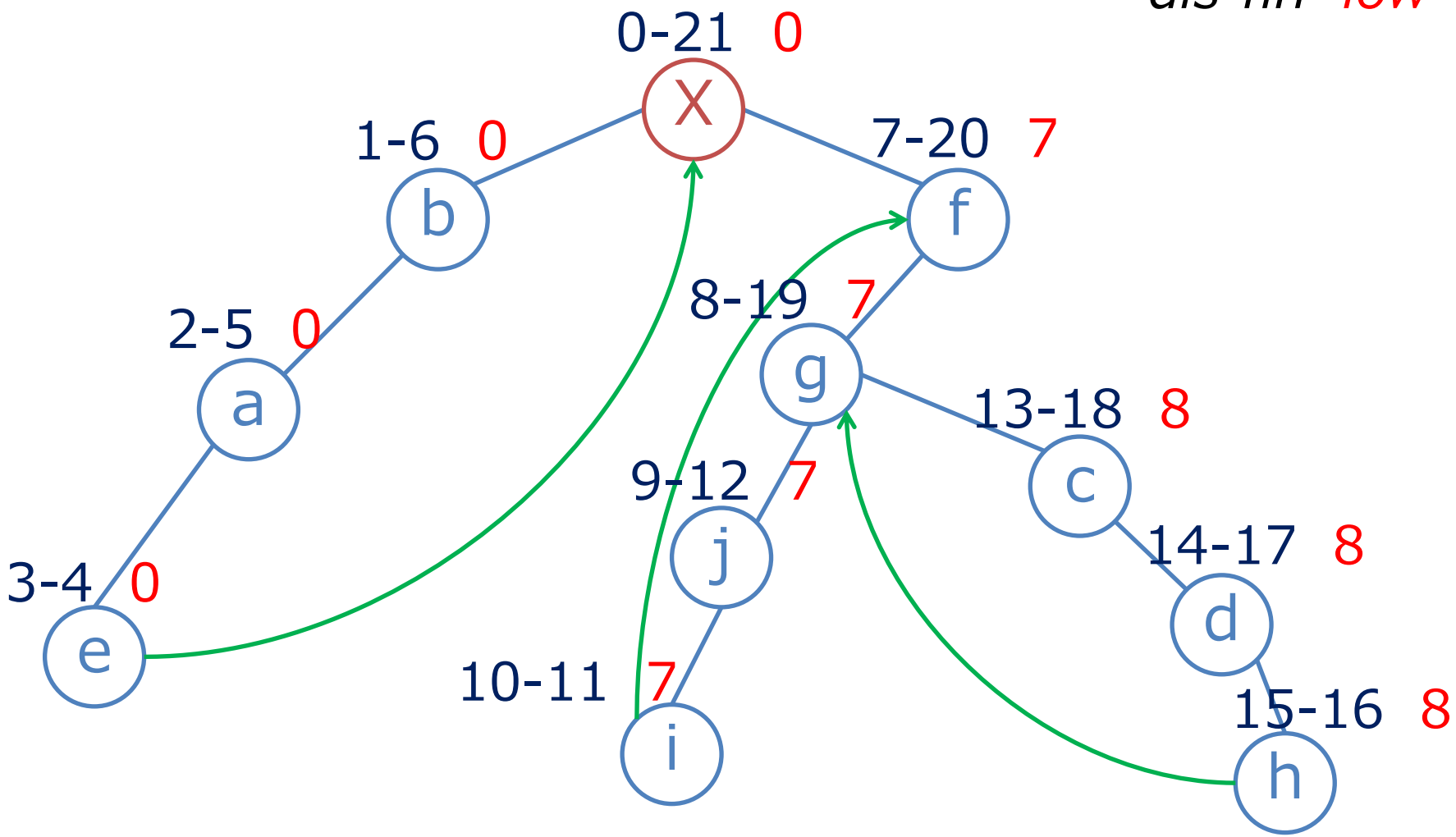
low[*u*] := min(*low*[*u*], *low*[*v*]).

else if (*v* が *u* の親でない):

low[*u*] := min(*low*[*u*], *dis*[*v*]).

fin[*u*] := *k*,　　*k* := *k* + 1.

DFS 木


dis-fin low
0-21 0
X
1-6 0
b
7-20 7
f
2-5 0
a
8-19 7
g
13-18 8
c
9-12 7
j
14-17 8
d
3-4 0
e
10-11 7
i
15-16 8
h

便利関数

頂点 v が頂点 u の子孫か？

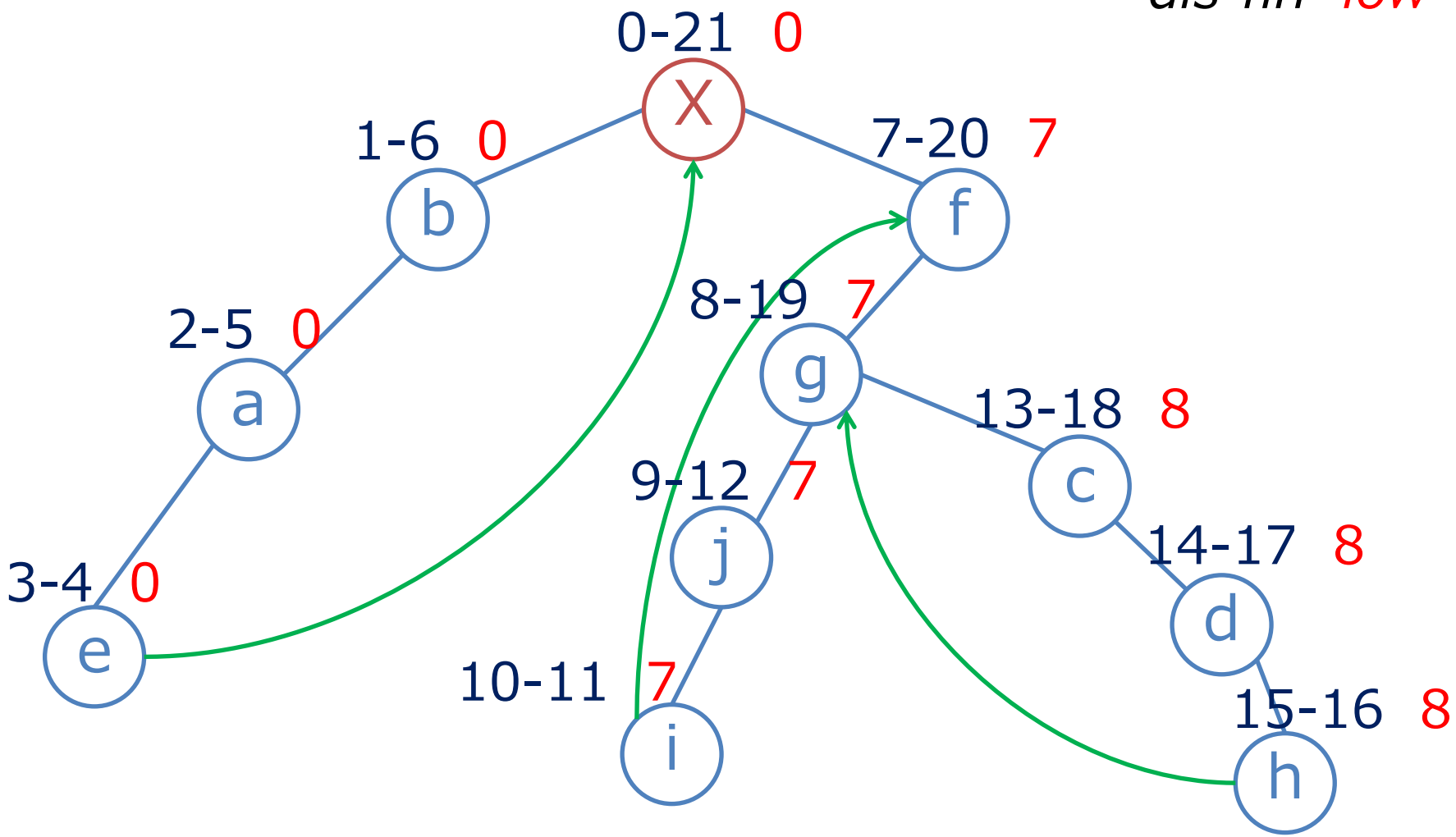
- $dis[u] \leqq dis[v]$ かつ $fin[v] \leqq fin[u]$

便利関数

頂点 v が頂点 u の子孫かつ $u \neq v$ のとき，u の子 w であって v が w の子孫であるものは？

- u の子すべてに対し先程の関数を使う
  - O(u の次数) (今回はこれで OK)
  - 二分探索すると O(log (u の次数)) も可能

DFS 木


dis-fin low
0-21 0
X
1-6 0
b
7-20 7
f
2-5 0
a
8-19 7
g
13-18 8
c
9-12 7
j
14-17 8
d
3-4 0
e
10-11 7
i
15-16 8
h

接続関係

頂点 u を取り除いたときに頂点 a と b は同じ連結成分に属するか？

Case 1. a も b も u の子孫でないとき

- Yes
  - 影響がない

接続関係

Case 2. a は u の子孫で，b は u の子孫でないとき

- a を子孫にもつ u の子を v_a とする
- $low[v_a] < dis[u]$ なら Yes
  - a が v_a 側から脱出できる

接続関係

Case 3. a も b も u の子孫であるとき

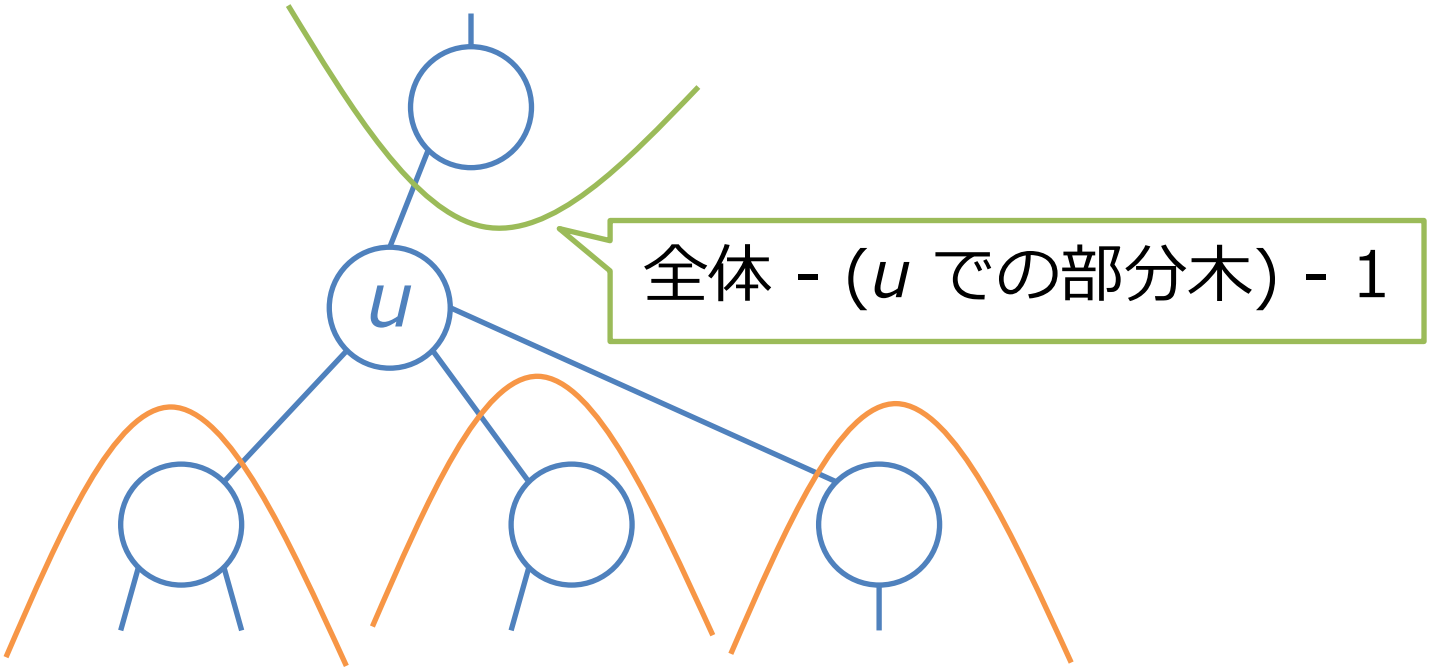
- a, b を子孫にもつ u の子を v_a, v_b とする
- 次のいずれかなら Yes
  - $v_a = v_b$
    - 脱出の必要なし
  - $low[v_a] < dis[u]$ かつ $low[v_b] < dis[u]$
    - a が v_a 側から，b が v_b 側から脱出できる

接続関係

- 箱の周囲 4 マスの接続関係
  - グラフの各頂点 u に対してそれを取り除いたときの隣接頂点たちの接続関係
  - 4 頂点の各組に対して同じ連結成分に属するか判定すればよい

連結成分のサイズ

- 各部分木のサイズを計算しておけばよい
  - *u* に木の辺で繋がっている頂点について部分木のサイズを足す



解法

- 状態：箱のあるマスと人のいる連結成分
- 遷移：箱の引っぱり方 (高々 4 通り)

- 各状態にたどり着くたびに，連結成分のサイズを答に足す
- $O(MN)$
  - 100 点
  - 4^2 倍という定数には注意

注意点

- 「プレイヤーと箱はそれぞれ壁でなく目標地点と異なるマスに配置しなければならず，プレイヤーと箱は異なるマスに配置しなければならない」
  - 箱が目標地点にいるときは答に足さない
  - 連結成分のサイズに目標地点は含めない
    - 目標地点を根にしておくとこれが楽

注意点

- *MN* 頂点のグラフに対して DFS を行う
  - スタックオーバーフロー
    - 今回のジャッジ環境では問題なかった
    - 手元で実行して怪しい挙動をしたときに焦らない
  - 自前のスタックを用意すれば再帰を避けられる
    - やや面倒

おまけ

- 倉庫番 (解く方) は一般には箱がたくさん
  - 盤面の状態が入力サイズの指数個
- **NP** に属するかどうかは未解決
  - 最短手数が入力サイズの指数になる場合あり
- **PSPACE** 完全であることが知られている
  - 入力サイズの多項式のメモリで解ける
    - そのような問題のなかで最も難しい
  - 参考：グラフの *s*-*t* 連結性判定，対数メモリ

得点分布