



プログラミング勉強会 - アルゴリズム編

第2回 ゲームで使うアルゴリズム

今日やること

- 前回の補足
- テトリスのピースの回転
- DFS (ぶよぶよのくっつき具合判定)
- BFS系 (最短距離を調べる)
- アクションゲームでの斜め移動

Vector? List?

■ vector

- ランダムアクセスが出来る(好きな要素を見れる)
- 挿入、削除が遅い(サイズ変更が遅い)

■ List

- ランダムアクセスできない(先頭、後ろから辿る)
- 挿入、削除が早い(サイズ変更が早い)

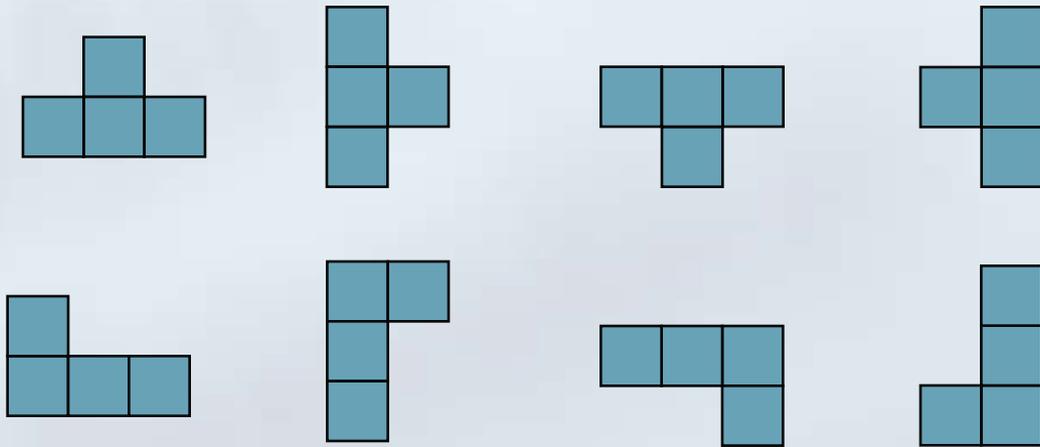
■ 基本はvectorでOK

■ サイズが頻繁に変わるならlist

ソート補足

- `sort` (VC++)
 - クイックソート + ヒープソート + 挿入ソート
 - 配列のサイズで変えている
- `stable_sort`
 - 追加領域を使わない $\rightarrow O(n(\log n)^2)$
 - 追加領域を使う $\rightarrow O(n \log n)$
- `stable_sort`をするよりも、
比較関数を用意して`sort`した方が速い。

テトリスのピース回転



最初から全パターン用意しとくのもあり。(面倒)
もっと楽にやるにはどうしたらいいか。

テトリスのピースの回転

- 行列を使って求める

$$\begin{pmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} -Y \\ X \end{pmatrix}$$

- 複素数を使って求める

$$(X + Yi) = (X + Yi) * i = (-Y + Xi)$$

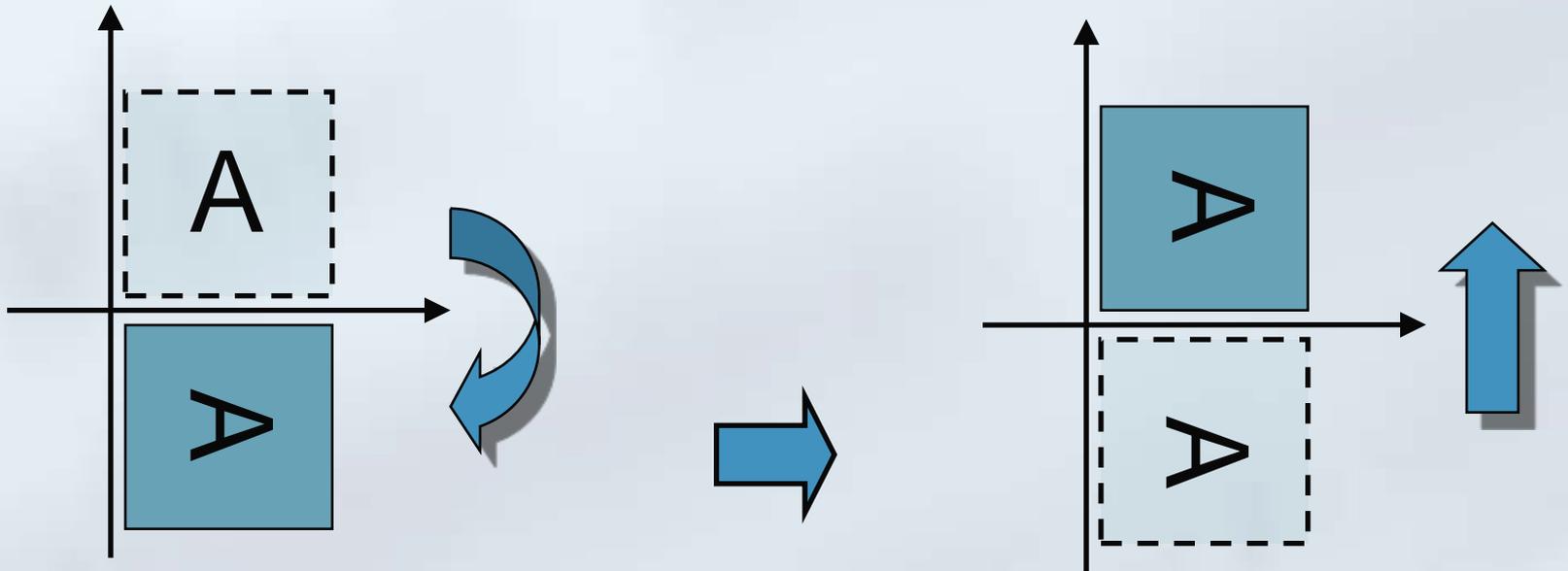
$$(X, Y) \text{ を } 90 \text{ 度回転} \rightarrow (-Y, X)$$

Rotate90.cpp

- rotate90A (そのまま書いた)
- rotate90B (少し効率がいい)

- $t[x][-y+(N-1)] = mat[y][x];$
 - $t[x][-y] = mat[y][x]$ じゃないの？
 - 前のページで求めたのは原点を中心にした回転

t[x][-y+(N-1)]の説明



回転させる

上にずらす

$$t[x][\underbrace{-y+(N-1)}_{\text{上にずらす}}] = \text{mat}[y][x];$$

上にずらす

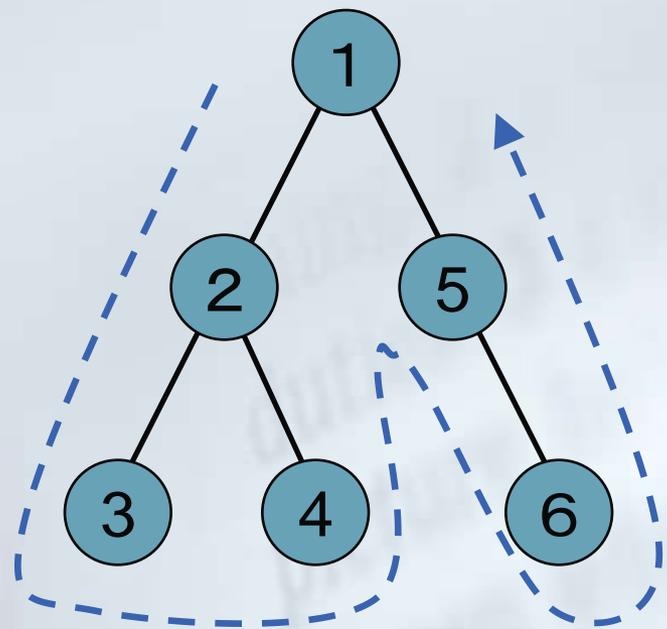
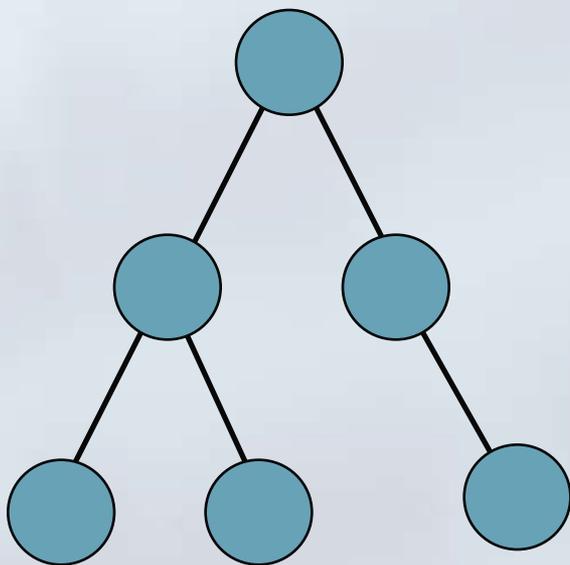
ここまでで何かわからない事、質問
確認したいことがあればどうぞ

グラフの話(というか主に探索)

- どんな時に使われるのか？
 - 探索
 - グループ分け(ぶよが何個くっついているかなど)
 - 最短距離、経路を調べる
 - 将棋、オセロなど、CPUの先読み

DFS

- Depth First Search (深さ優先探索)
- 行けるだけ先に進む。進めなかったら戻る
- 全探索などを簡潔に書ける

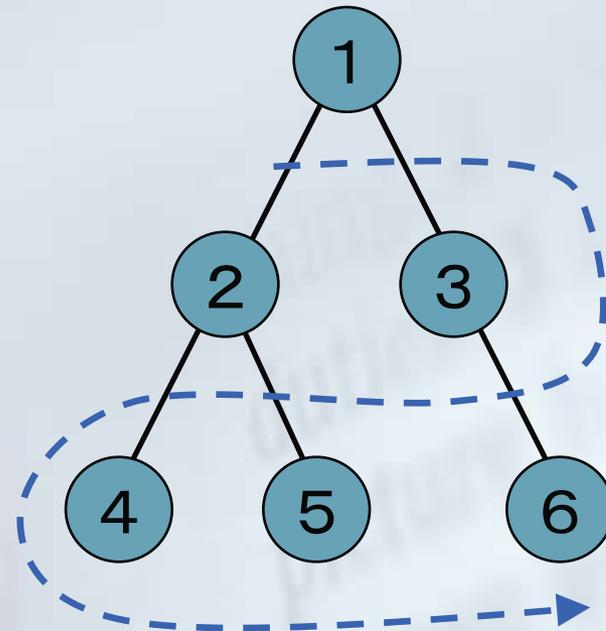
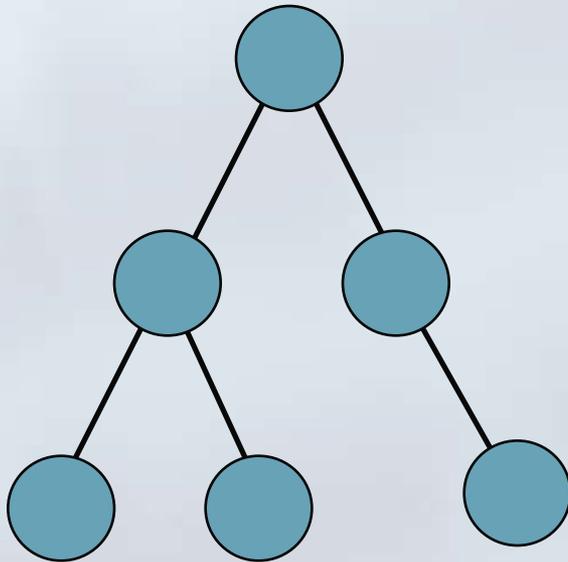


例：ぷよぷよのくっつき具合判定

- Puyo.cpp
- countPuyoで再帰的に求めている
- コメントにほとんど書いてある

BFS

- Breath First Search (幅優先探索)
- 同じ高さを全て見てから次の高さに行く
- 最短経路が求まるという性質がある



例：SからTまでの最短距離（BFSで）

									17			
	10	9	8	9	10	11	12	13	T			
		8	7		11	10		14	15	16	17	
			6			9		15	14			
	5	4	5	6	7	8	9		13			
		3		5	6	7	8		12	13	14	15
	1	2	3	4	5				11			16
0	S			5	6	7	8	9	10			17

（どこから来たかを保存すれば経路も求まる）

実装の話 (BFS.cpp)

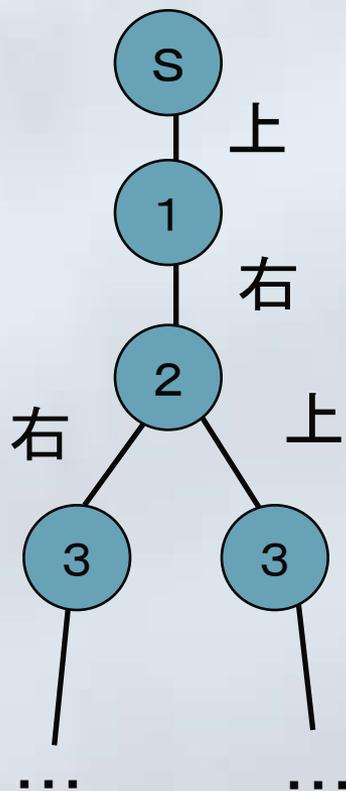
- キューを使う
 1. スタート位置をキューに入れる
 2. キューの先頭から1つ取り出す(それをXとする)
 3. Xが目的地なら終了
 4. Xから行ける所をキューに入れる
 5. 2に戻る
- 途中でキューが空になったら目的地にたどり着けなかったという事になる

#include <queue>

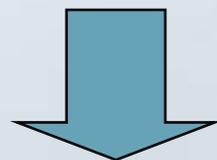
- キューがすでに用意されている
- 使い方 (Queue.cpp)
 - `queue<型>` で宣言
 - サイズ知りたかったら `size()`
 - 空かどうか知りたかったら `empty()`
 - 後ろから要素を入れるには `push()`
 - 先頭の要素を見るには `front()`
 - 先頭から取り除くには `pop()` ※取り除くだけ

グラフ理論の便利さ

前のページだと左下のようなグラフになる



グラフに落とせるものだったら
何にでも応用できる

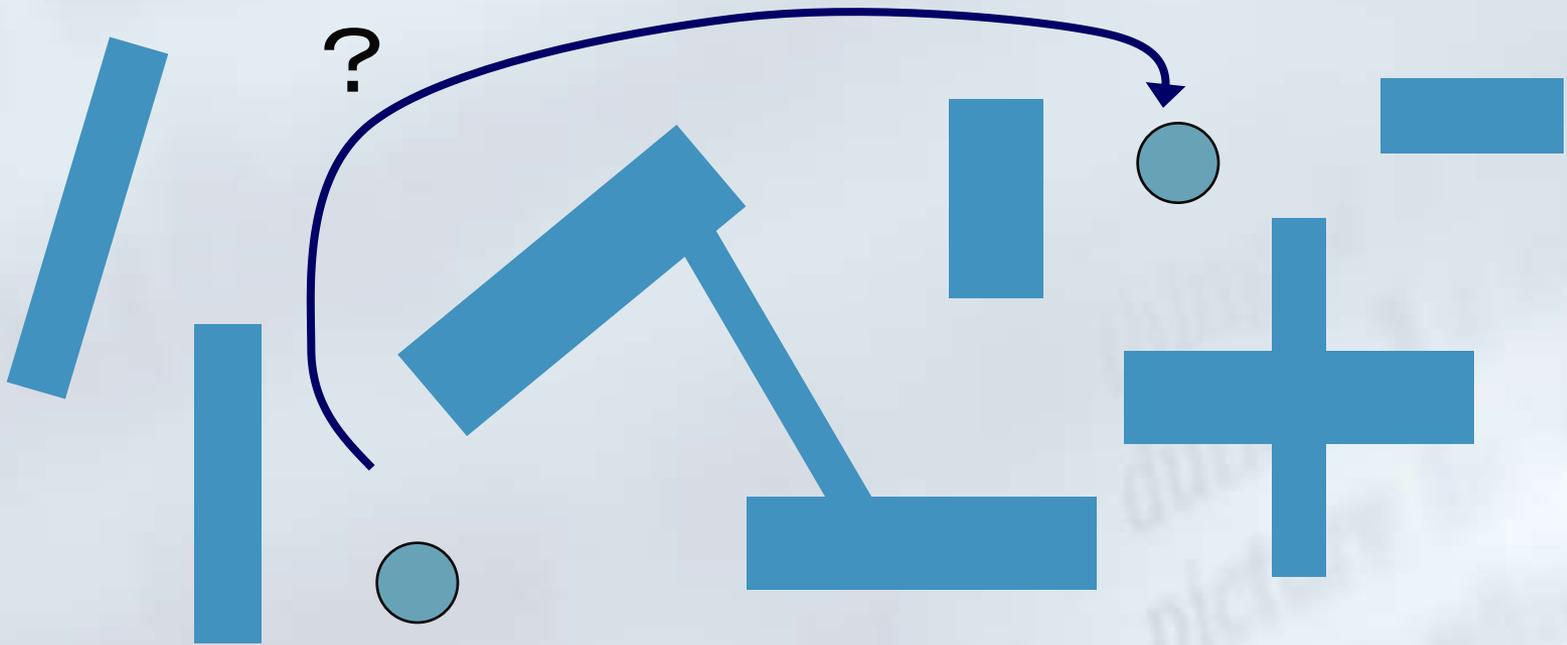


応用範囲がすごく広い

ここまでで何かわからない事、質問
確認したいことがあればどうぞ

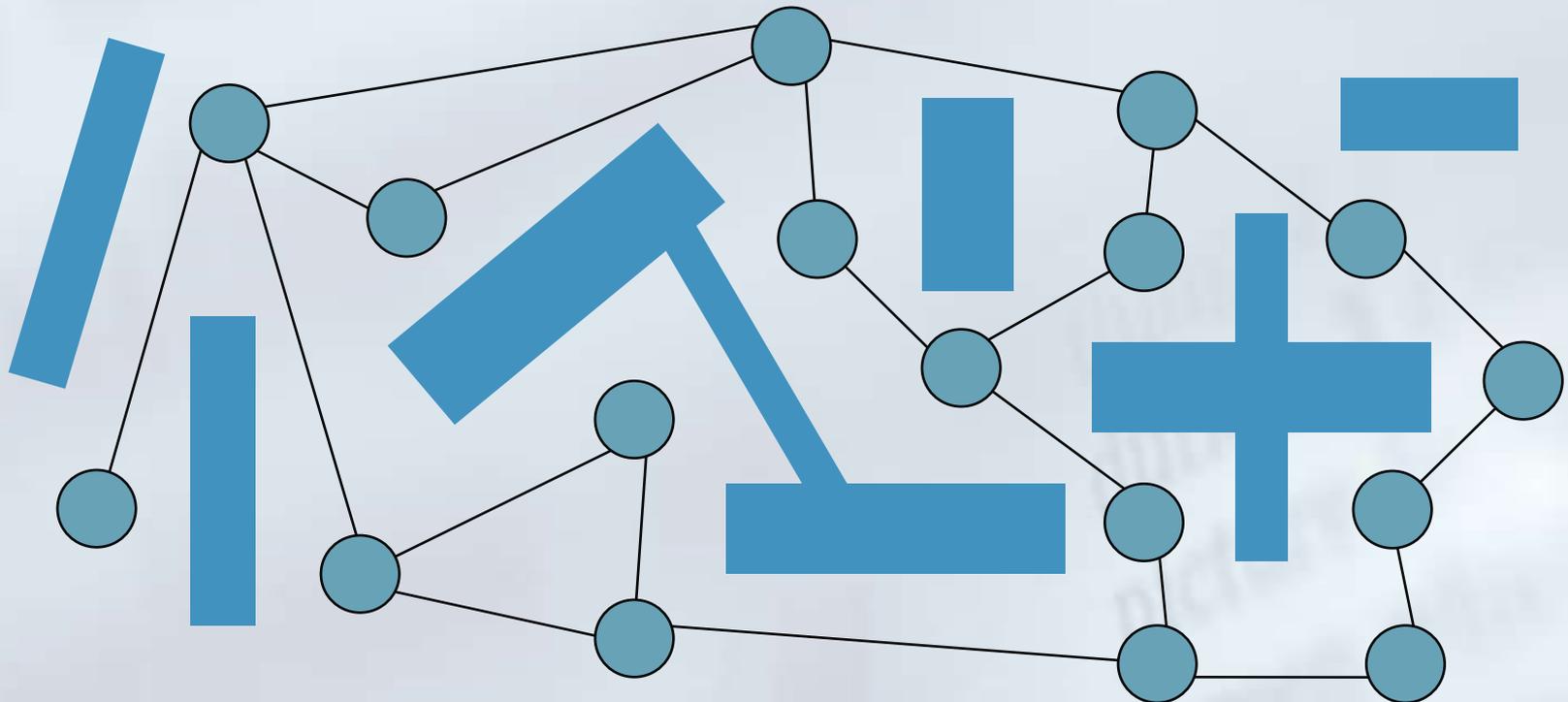
一番短い経路

- 迷路というか下のような障害物の置いてあるマップがあって自由に動けるとする。で、ある点からある点の最短の行き方を調べたい。



グラフに落とす

- なにか迷路っぽいのがあって、ある点からある点までを最短で移動したいとかそういう時は、基準点を作ってそれを繋ぐとグラフになる

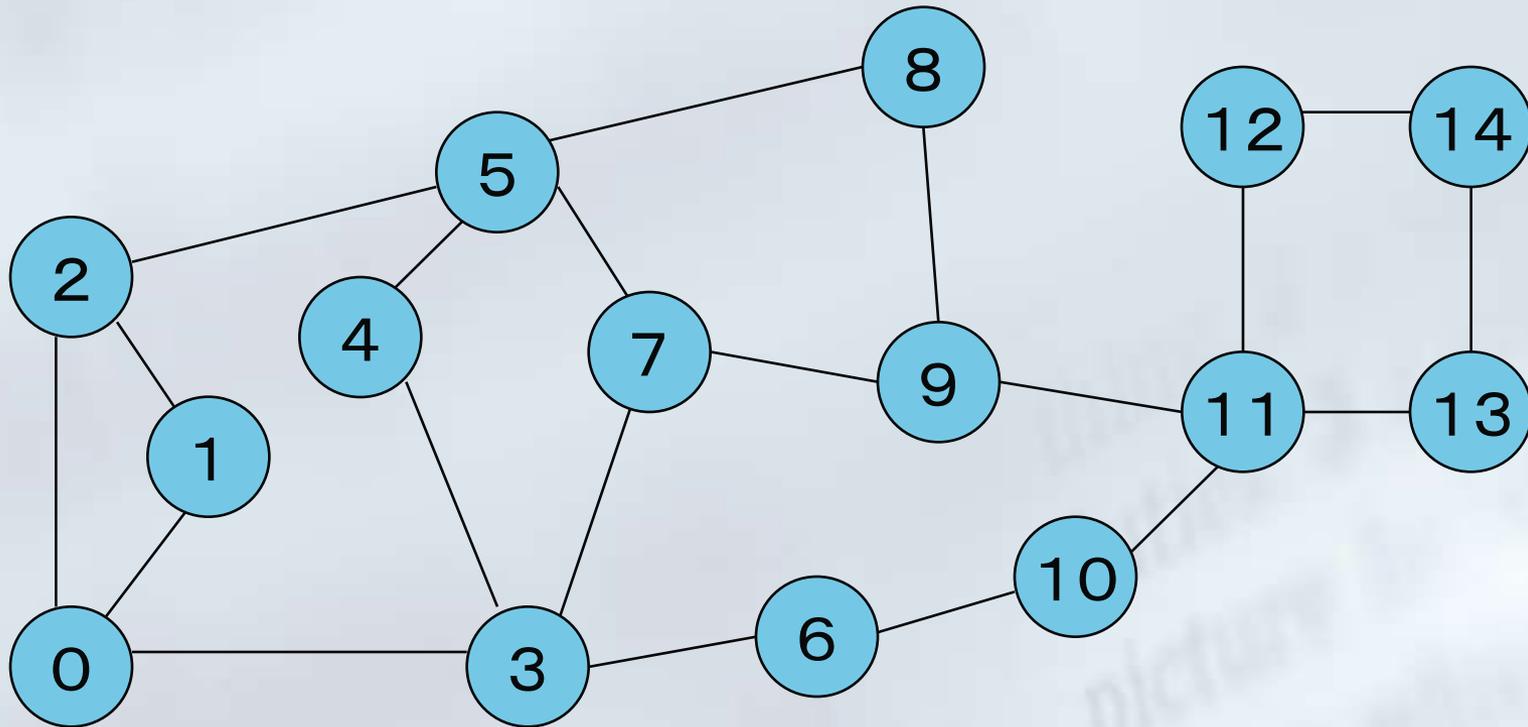


ダイクストラ、A*

- 少し応用的なアルゴリズム
- ダイクストラ
 - 最短経路をBFSよりも速く求める
 - やり方は最小全域木を作る時の動きに似ている
 - 厳密な解が求まる
- A*
 - ダイクストラ(というか順位優先探索)にヒューリスティクス(近似的)を加えて速度アップしたようなもの。

ダイクストラ (Dijkstra.cpp)

- `priority_queue`を使う
- コスト(距離)が一番低いノードから順に見ていく



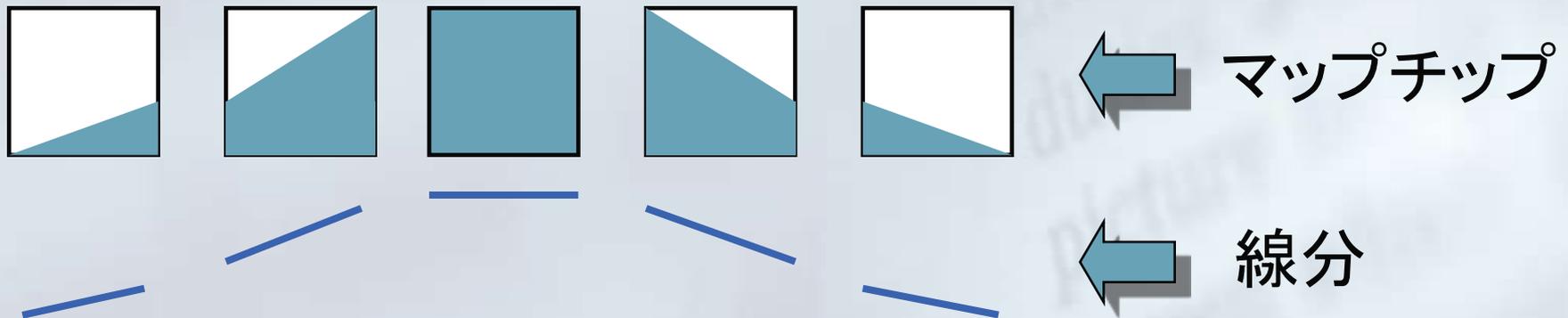
A* (AStar.cpp)

- ダイクストラに速度をプラス
- しかし正確に求められないこともありうる
- 今回はノードを選ぶ時に「コスト+ゴールまでの距離」が一番低いものから選んでいく
 - ダイクストラの順序のつけ方が変わっただけ
- ゲームで使うならこっちの方が向いている気がする

ここまでで何かわからない事、質問
確認したいことがあればどうぞ

斜め移動のやり方

- 高さをどうやって表現するか
- やり方はいくつかもある
 - それぞれのマップチップにこの位置だとこの高さというテーブルを作っておく
 - マップチップではなく線分として判定(現実的)
- どうやって角度を求めるか



実装

- おそらく次回の方が内容的に合っているので詳しくは次回。
- 時間があれば例プログラムと解説を軽くします

まとめ

- テトリスのピースの回転は簡単に書ける
- ふよふよのくっつき具合も簡単に調べれる
- DFSの再帰、BFS系はキューを使って実装する
- 最短経路を求めるにはいろいろな方法がある
- グラフ理論は応用の幅が広い

全体を通して何か質問、
確認したいことなどがあればどうぞ

おすすめ参考図書、参考ワード

- アルゴリズムイントロダクション1～3巻
- グラフ理論入門 (R.J.ウィルソン)
- グラフ理論についての参考ワード
 - トポロジカルソート
 - 反復深化
 - 双方向BFS
 - ベルマンフォード
 - ワーシャルフロイド
 - ミニマックス法
 - $\alpha\beta$ 法
 - 最大フロー
 - 最小全域木
 - バックトラック

次回予告

- ジオメトリ(平面幾何学)
- 複素数

- 具体的には・・・
 - 複素数は便利って話
 - ICPC用ジオメトリライブラリの公開と解説
 - 線分とかと線分とかの交差判定(スマートに)
 - 線分とかと線分とかの距離(スマートに)
 - 3次元幾何学は考え中

終わり

おつかれさまでした