限界！アルゴリズム　後退解析

24. JUNE 2023

Created by ももも @momomorgentau

はじめに

AtCoderの問題で時々ありますよね

高橋君と青木君が対決するやつ

今日は対決シリーズのうちの１つである後退解析というアルゴリズムについてお話していきます。 これ系の問題で大変困るのが

最善な行動

という悪魔の言葉です。

この点に関して詳しく説明出来ればと思います。

アジェンダ

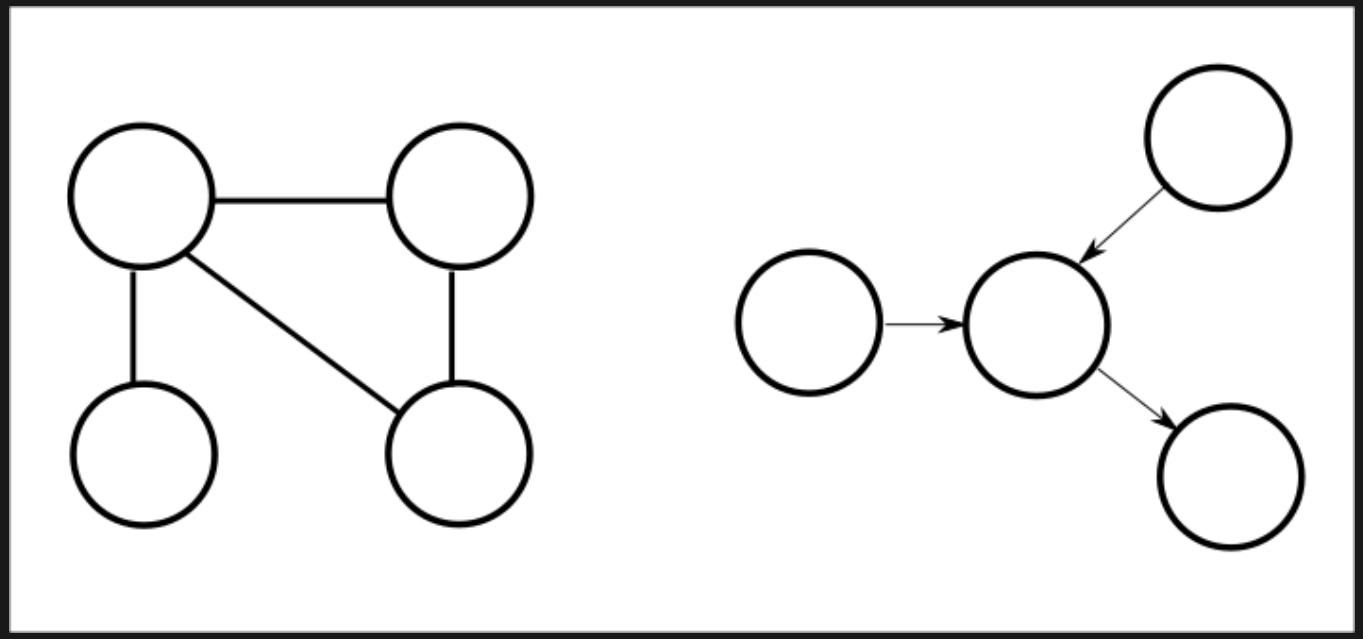
グラフのいろは
背景説明
問題紹介
アルゴリズム説明
問題解説
おわりに

グラフのいろは

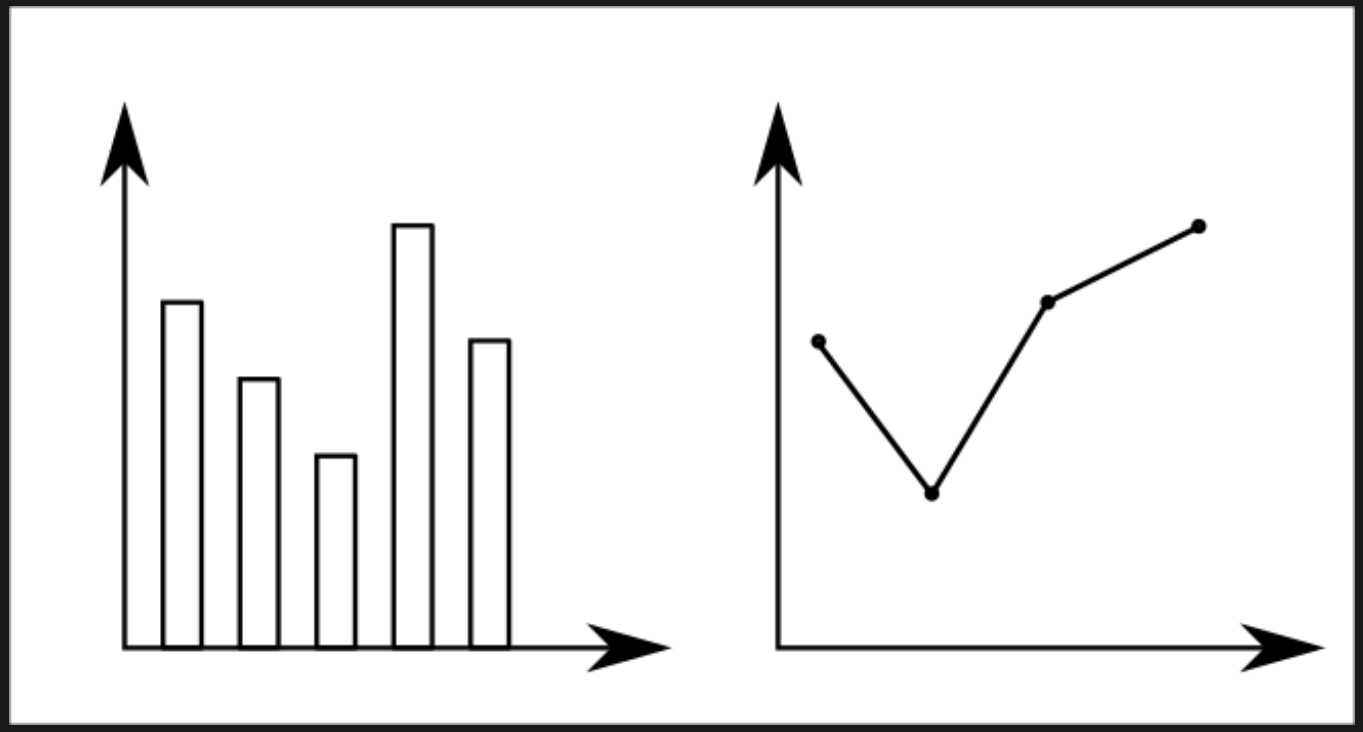
後退解析の前に、、
グラフに関する基本的な説明と実装について
説明していきます

グラフって？

〇とーで構成されたやつです。ーじゃなくて→の場合もあります



一般的には以下のようなものもグラフと呼ばれます



ですが、競技プログラミングの文脈では、

先ほどの画像のようなものを指します

グラフについて次の3つを抑えていただきたいです
といいますか、これだけ分かってれば色々な問題が解けます

- グラフの種類
- プログラムの表現方法
- 探索方法

グラフの種類

木、完全グラフ、スターグラフ、、などなど、
あげればキリがありません
ですが、本日覚えていただきたいのは次の２つです

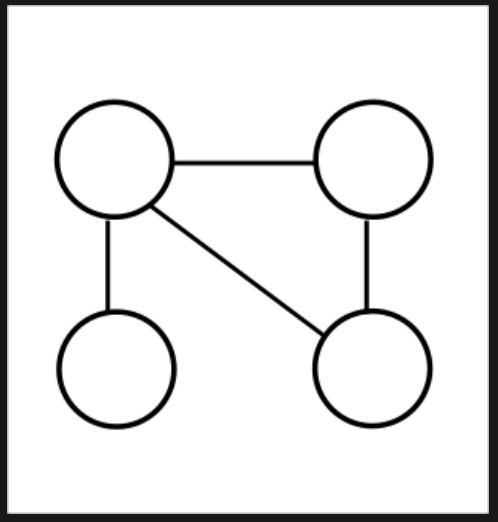
- 無向グラフ
- 有向グラフ

無向グラフ

読んで字の如く

方向の指定が無い

グラフです。ーで繋がっているならどこへでも行けちゃいます

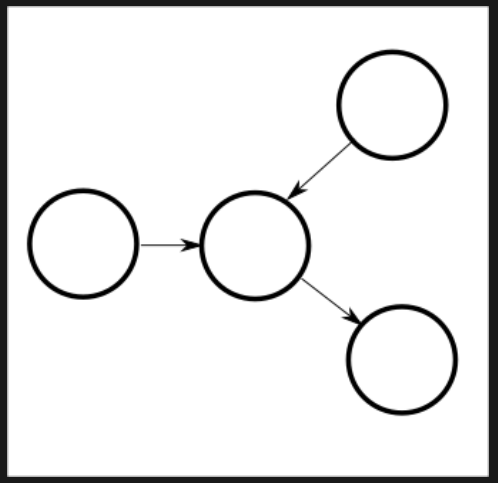


有向グラフ

一方でこちらは

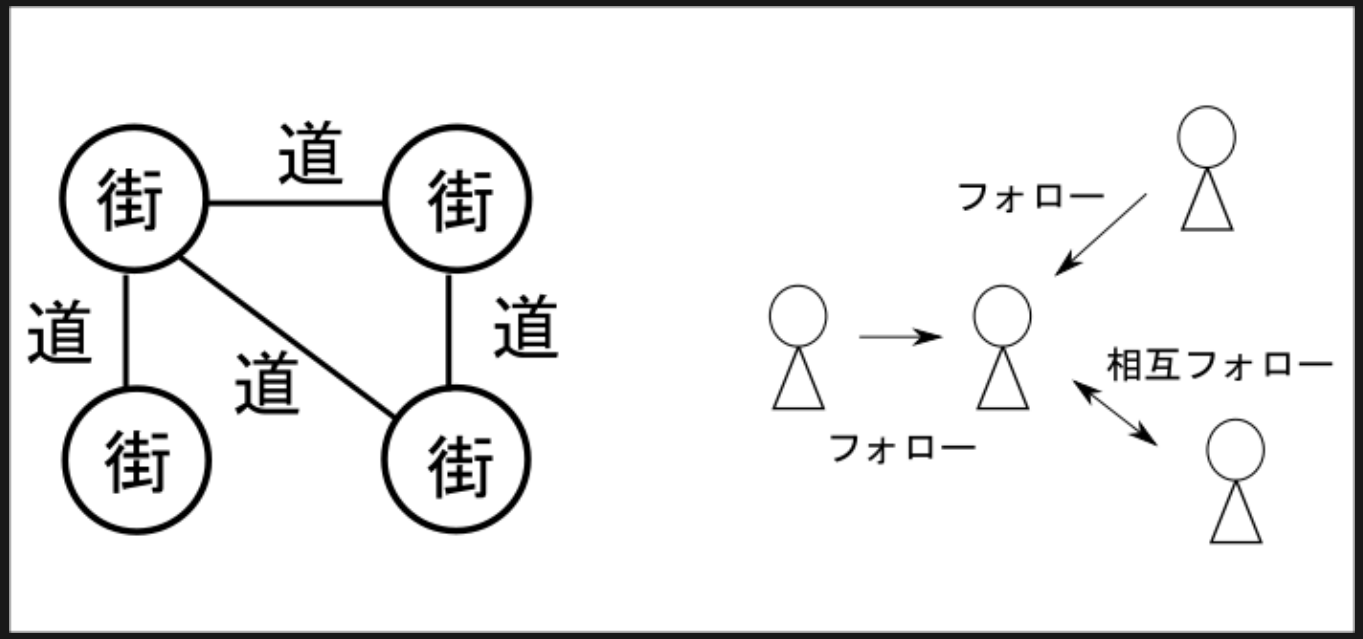
方向の指定が有る

グラフです。一方通行ってやつです。



グラフの例

こんな感じでグラフは使われます



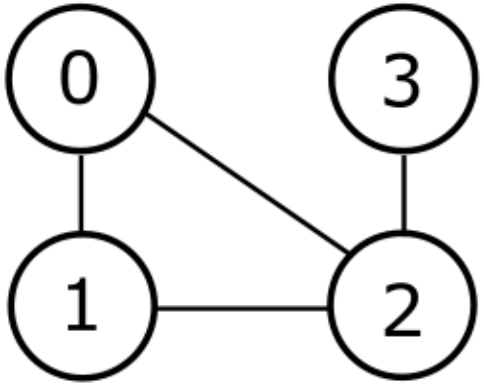
プログラムの表現方法

グラフをプログラムで表現する方法は主に２つです

- 隣接行列
- 隣接リスト

隣接行列

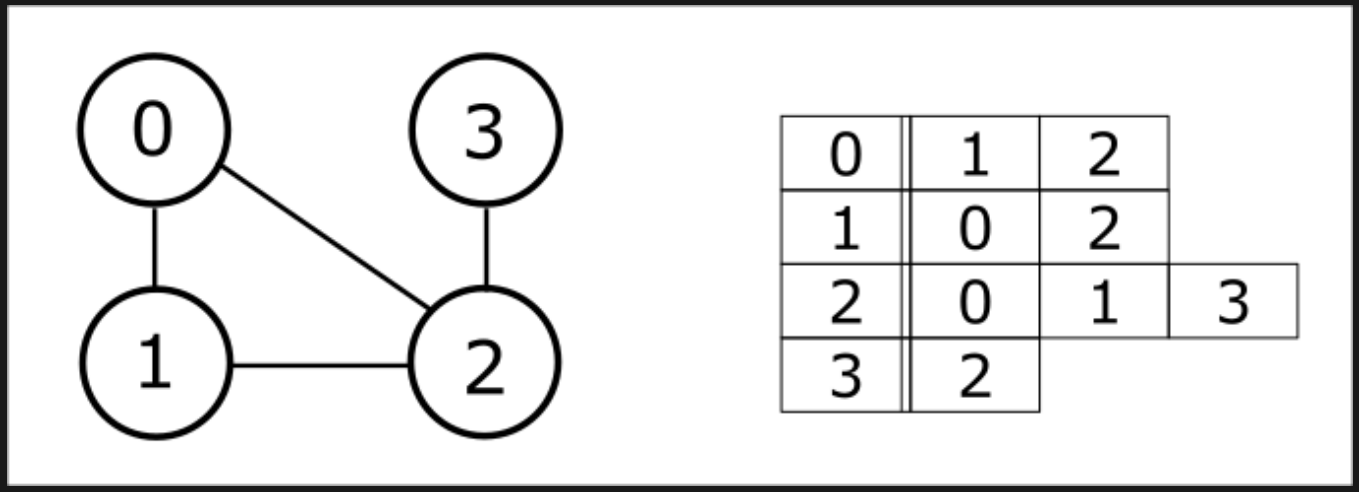
各頂点間の辺の情報を２次元配列で表したもの



	0	1	2	3
0	false	true	true	false
1	true	false	true	false
2	true	true	false	true
3	false	false	true	false

隣接リスト

頂点ごとに辺の情報をまとめた配列



隣接行列・隣接リストの実装

- C++
- python

隣接行列 vs. 隣接リスト

圧倒的に 隣接リスト の使用頻度の方が多いです

- 少ない情報で管理可能
  隣接行列 頂点数 * 頂点数 を管理する
  隣接リスト 辺の数分管理する

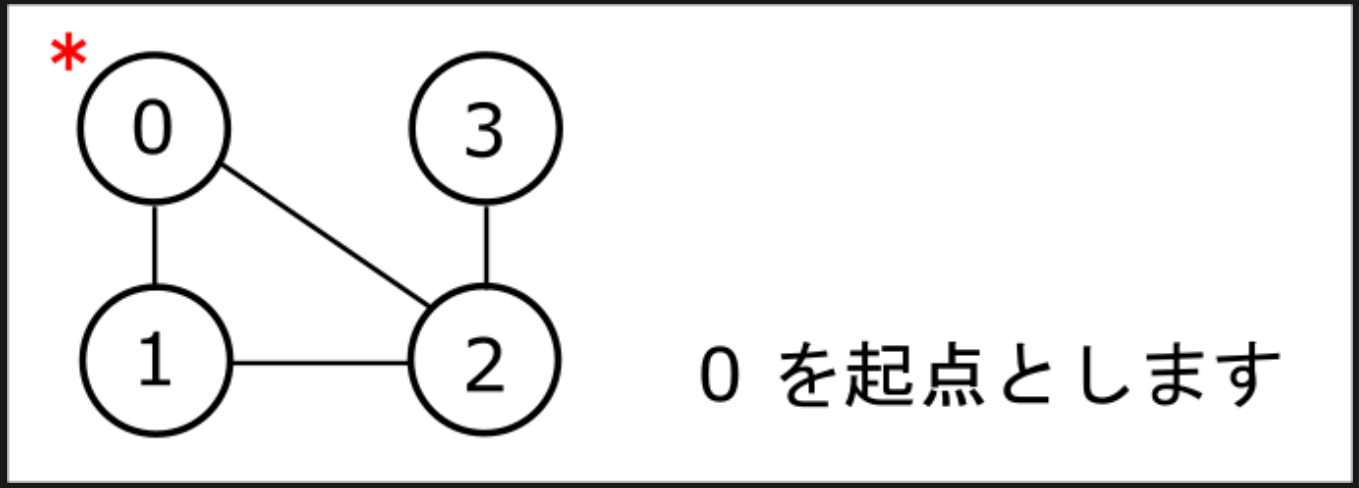
  頂点数 $5 * 10^5$ の問題が多いため
  これを2乗するととんでもない数になってしまいます

探索方法

隣接リストにて作成したグラフを探索していきます
探索は主に次の２つの作業の繰返しで概ね大丈夫です

- 起点となる頂点を決定する
- 起点に繋がっている頂点に対して処理を行う

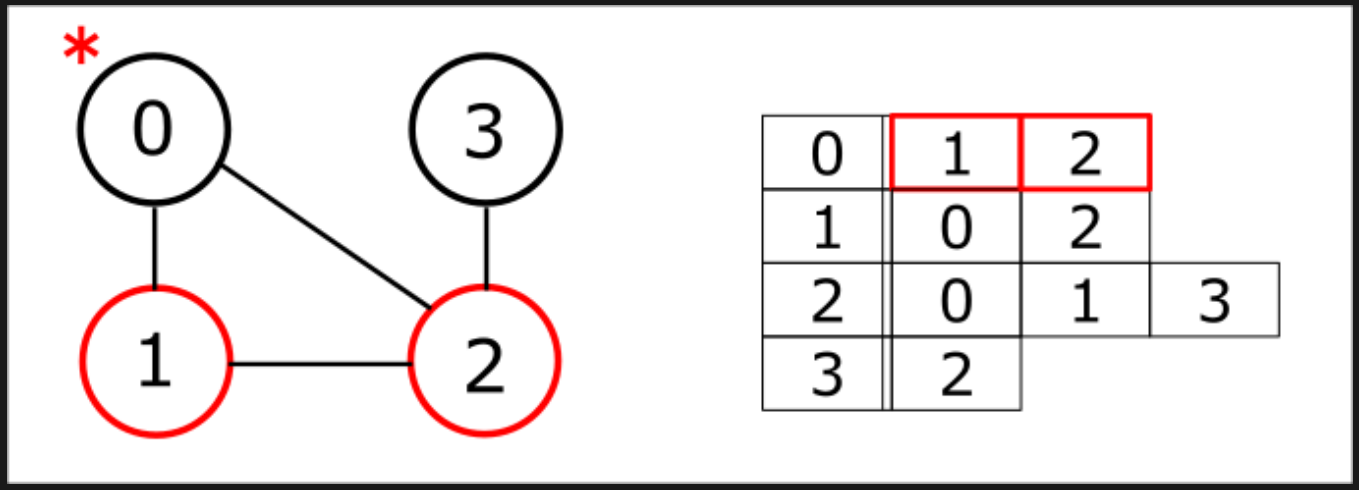
先ほどのグラフで考えます



開始点は適当に 0 としておきましょう

これで起点の頂点が決まりました

起点に繋がっている頂点を見ていきましょう



0	1	2	
1	0	2	
2	0	1	3
3	2		

プログラムでは以下のように表現されます

- C++
- python

foreach文と言うとピンとくる方がいるかもしれません

処理が終わったらが次の起点を決めていきます
ここでの決め方、タイミングは探索方法によって変わってきます
（探索方法によっては開始点の決定にも影響があります）

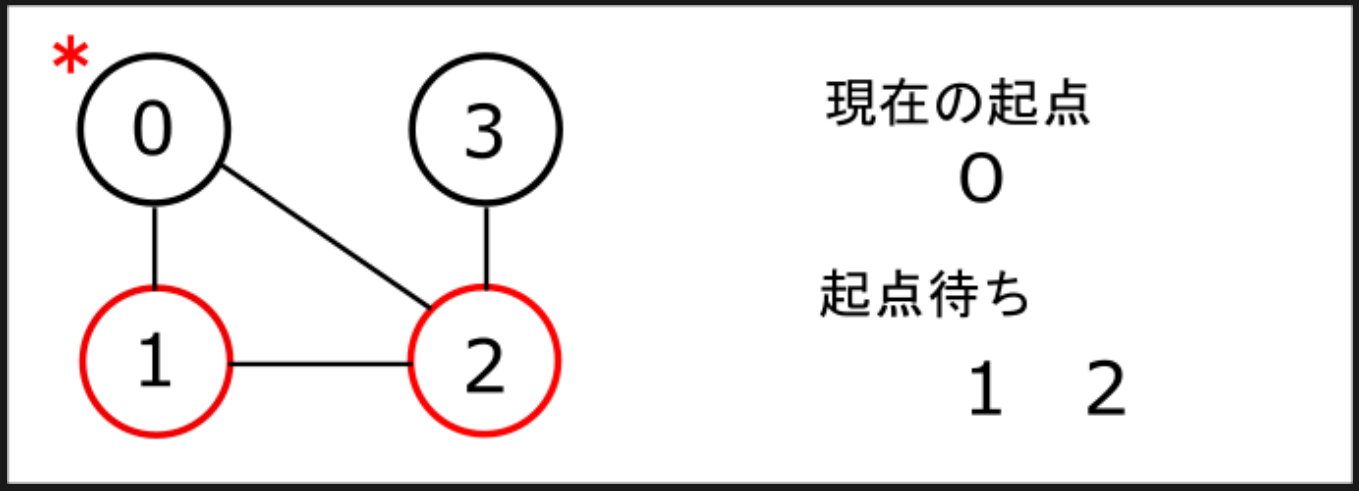
- 幅優先探索 BFS
- 深さ優先探索 DFS
- ダイクストラ法

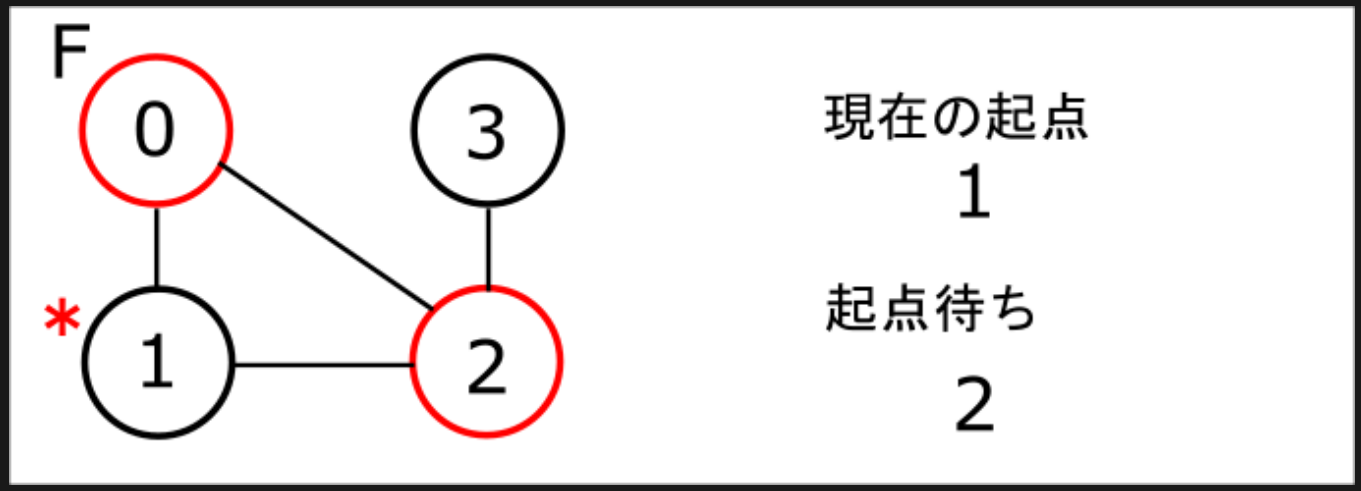
などなど

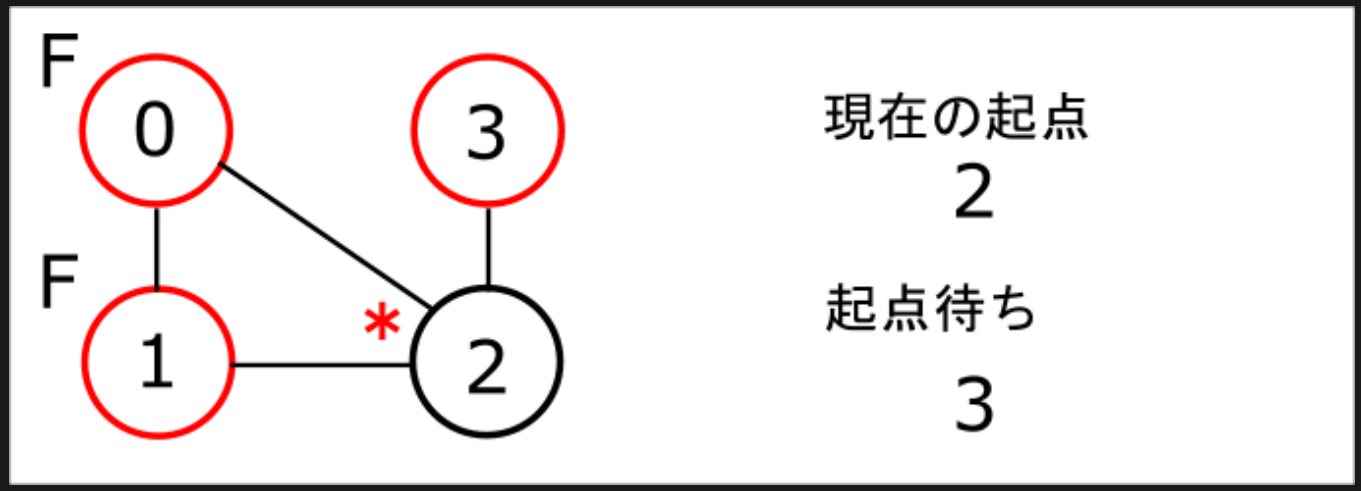
幅優先探索 BFS

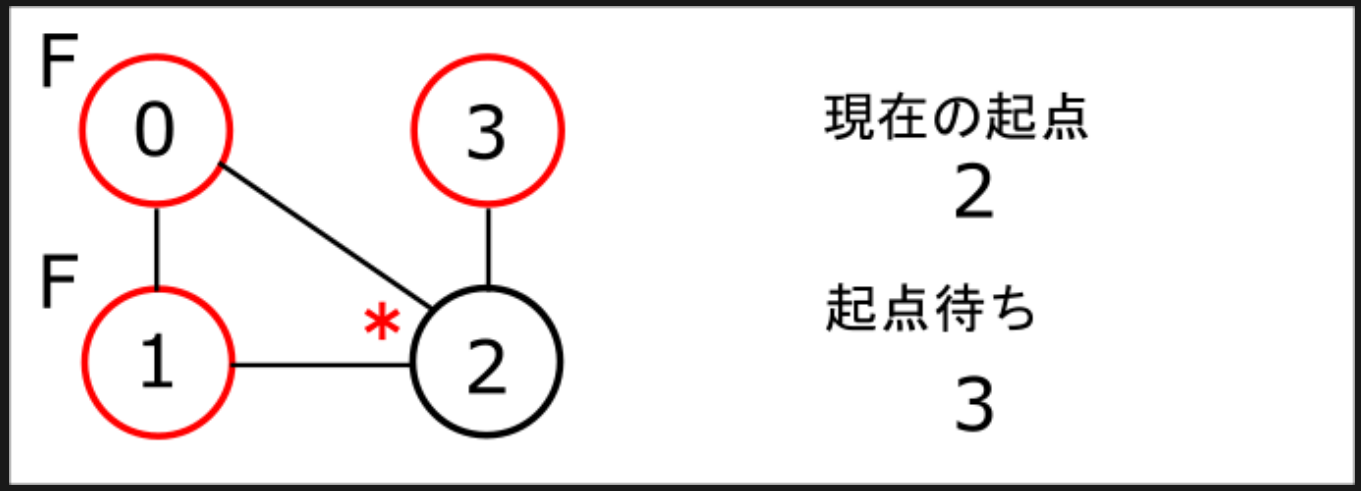
こいつは一番シンプルです

近いところから順番に探索していきます




F
0
3
*
1
2
現在の起点
1
起点待ち
2

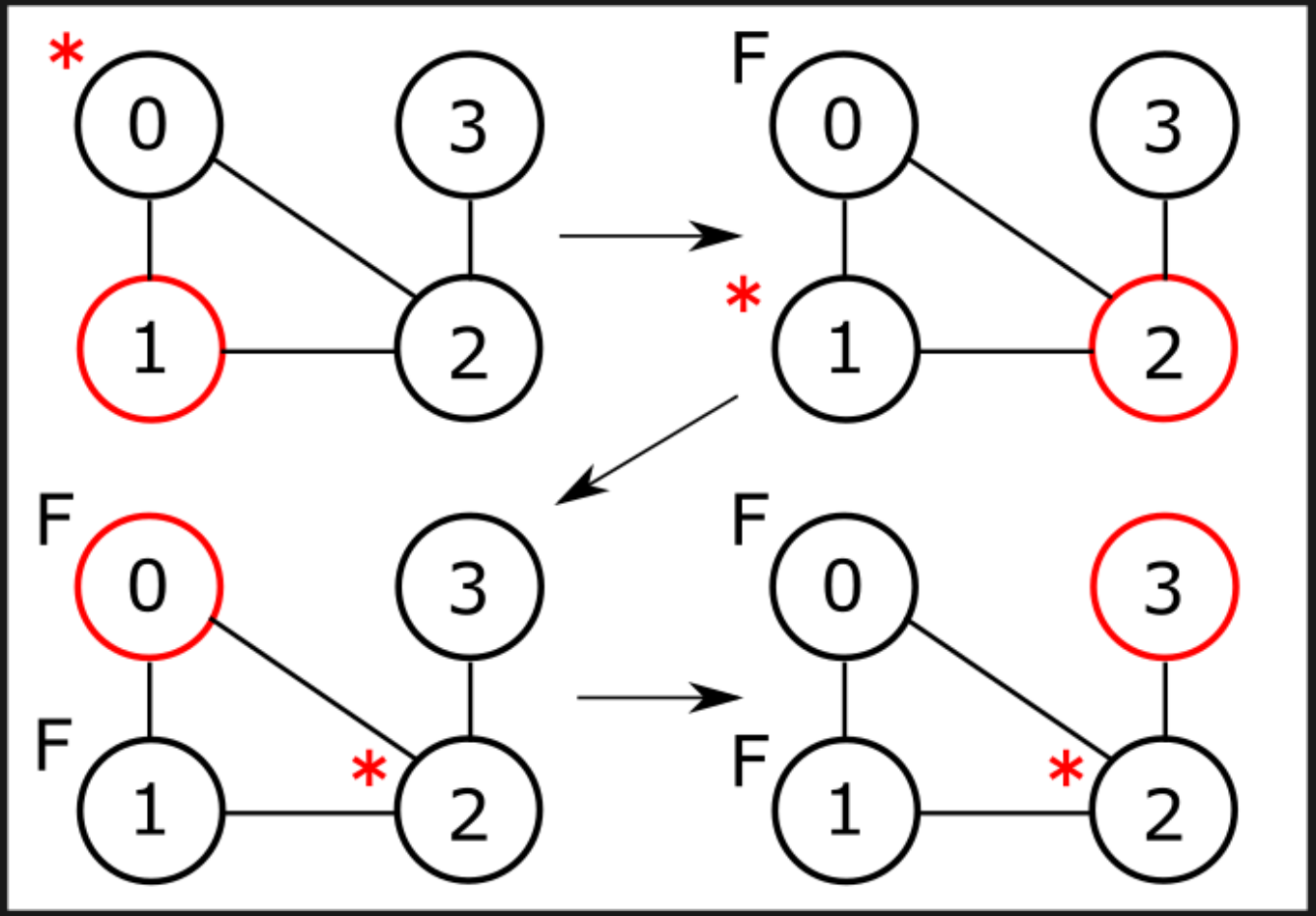

F
0
3
F
1
*
2
現在の起点
2
起点待ち
3


F
0
3
F
1
*
2
現在の起点
2
起点待ち
3

深さ優先探索 DFS

こいつは少し実装に癖があります（個人の感想）

とにかく先へ先へと探索していきます


*
0
3
1
2
F
0
3
*
1
2
F
F
0
3
F
*
1
2
F
F
0
3
F
*
1
2

ダイクストラ法

頂点間の最短経路に良く用いられるアルゴリズムです
現時点でコストが最小のものを起点にします
このとき繋がっていない頂点が次の起点になることもあります

図略

以上の方法により 最短経路 や 各種コスト を計算していきます

いくつか例題を掲載しておきます

- スプリンクラー（第3回 アルゴリズム実技検定 E問題）
  繋がっている頂点に対する操作を行います。まずはこの問題から
- Tour （ABC 204 C問題）
  計算量改善等が不要な純粋なグラフ探索です。BFS DFSの練習にどうぞ

以上でグラフの説明はおしまいです

ちなみに後退解析はまた別の探索方法になりますのでお楽しみに

背景説明

まずはアルゴリズムの説明を始める前に
近年耳にしない日はない
AI
に関して さらっとお話しさせてください。

近年の "AI" に関して

最近は"お絵かき"のAIが非常に注目されていますが、もう少し前に
遡ってみると 囲碁や将棋のAIが世間を騒がしていました。

囲碁 ALPHA GO (2015)

将棋 水匠 (2022)

これらのAIがプロの棋士と対決するニュースを目にします

将棋電王戦の結果は

14勝5敗1分

らしいです。(AI側の成績です)

確かに強いですが 絶対最強！ ではないですね。

将棋電王戦（しょうぎでんおうせん）*[注 1]*とは、ドワンゴが主催するプロ棋士とコンピュータ将棋ソフトウェアとの非公式棋戦である。 映像メディアが主催する棋戦としてニコニコ生放送による中継と、対局者やソフトウェア開発者などをフィーチャーした事前*PV*が特徴。 【*wikipedia*】

学習の問題点

複雑でパターンが非常に多い

将棋：10^{60} 10^{220}

詳しい方いらっしゃいましたら教えてください(文献)

AtCoderにて解いている問題

10^{7}、10^{8}

これで2秒ほど計算時間がかかるので、単純計算で、、、

とにかくとんでもない時間かかかります。

そのため、

理論的に絶対に勝てる選択

ではなくて、少しだけ妥協しまして

絶対ではないけど結構強いよ

という選択を学習していきます

ということで人間相手にも負けることがあるわけです

裏を返してみます

繰返しにはなりますが

学習の問題点

複雑でパターンが非常に多い

ということは、、

そんなに複雑でなくパターンが少ない

場合であれば、適切なアルゴリズムを使えば絶対に勝てる手を求められるということです。

適切なアルゴリズムって？

問題によって様々ありますが、本発表では

後退解析

を紹介していきます。通常の将棋を縮小した「どうぶつ将棋」の最適解を求められる方法としても知られています。

その他にも「Grundy数」も有名ですね。時々ABCでも出題されています。

問題紹介

本発表で扱う問題を紹介します。
聴講者がこの問題を解けるようになることが今回の目標です。 黄色の問題ですので心して聞いてください。

ABC209 E-Shiritori

高橋辞書には N 個の単語が載っており、 i ($1 \leq i \leq N$) 番目の単語は s_i です。
高橋君と青木君は高橋辞書を使って 3 しりとりをします。 3 しりとりのルールは以下です。

- 高橋君と青木君は、高橋君から始めて交互に単語を言い合っていく。
- 各プレーヤーは前の人が言った単語の最後の 3 文字で始まる単語を言わなければならない。 例えば、前の人が Takahashi と言った場合、次の人は ship、shield などを言うことができ、 Aoki、sing、his などを言うことはできない。
- 大文字と小文字は区別される。例えば Takahashi のあとに ShIp を言うことはできない。
- 言う単語がなくなったほうが負ける。
- 高橋辞書に載っていない単語をいうことはできない。
- 同じ単語は何度でも使ってよい。

各 i ($1 \leq i \leq N$) について高橋君が 3 しりとりを単語 s_i から始めたときどちらが勝つかを判定してください。ただし、二人とも最善に行動するとします。 具体的には、自分が負けないことを最優先し、その次に相手を負かすことを優先します。

制約

- N は 2×10^5 以下の整数
- s_i は英小文字と英大文字のみからなる長さ 3 以上 8 以下の文字列

出力

N 行出力せよ。 i ($1 \leq i \leq N$) 行目には、 3 しりとりを単語 s_i から始めたとき、 高橋くんが勝つなら Takahashi 、青木君が勝つなら Aoki 、しりとりが永遠に続き勝敗が決まらないなら Draw と出力せよ。

入力例を見ていきます

入力例 1

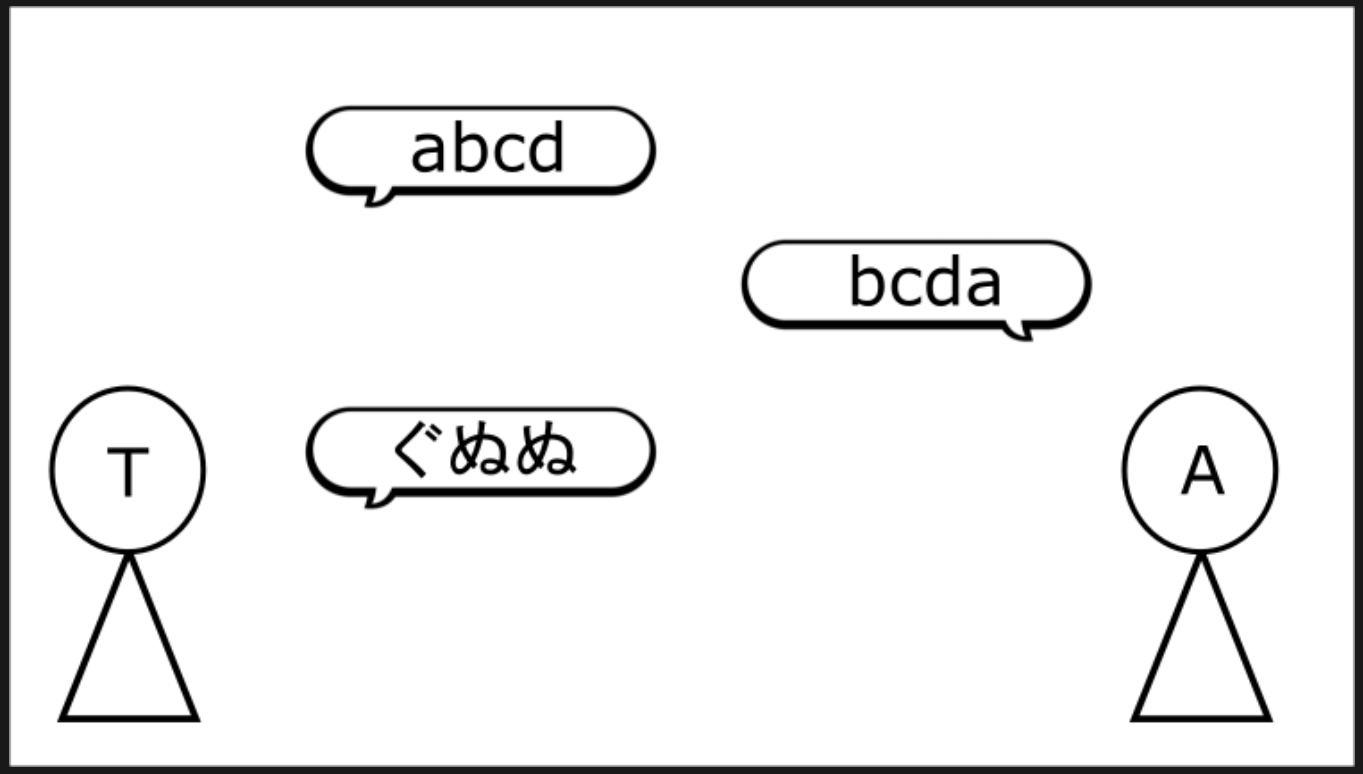
3

abcd

bcda

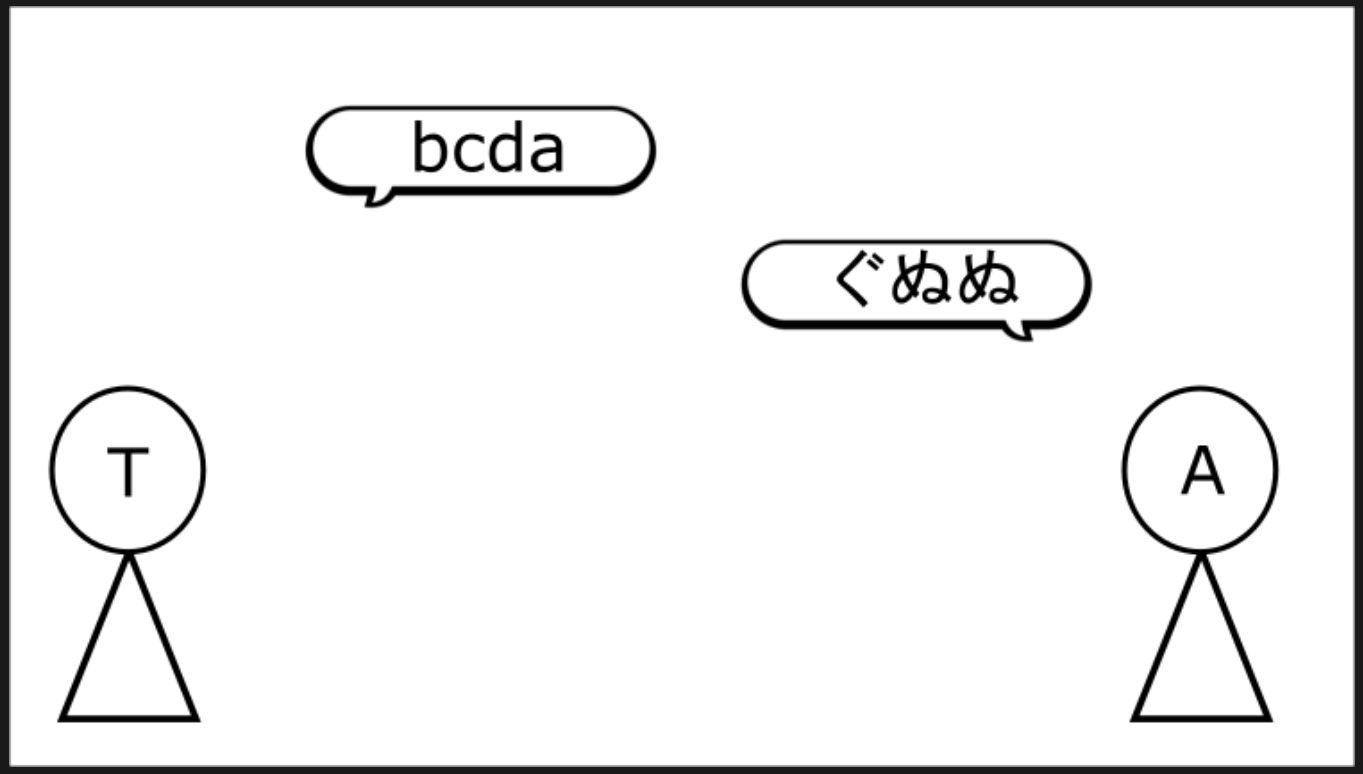
ada

abcd



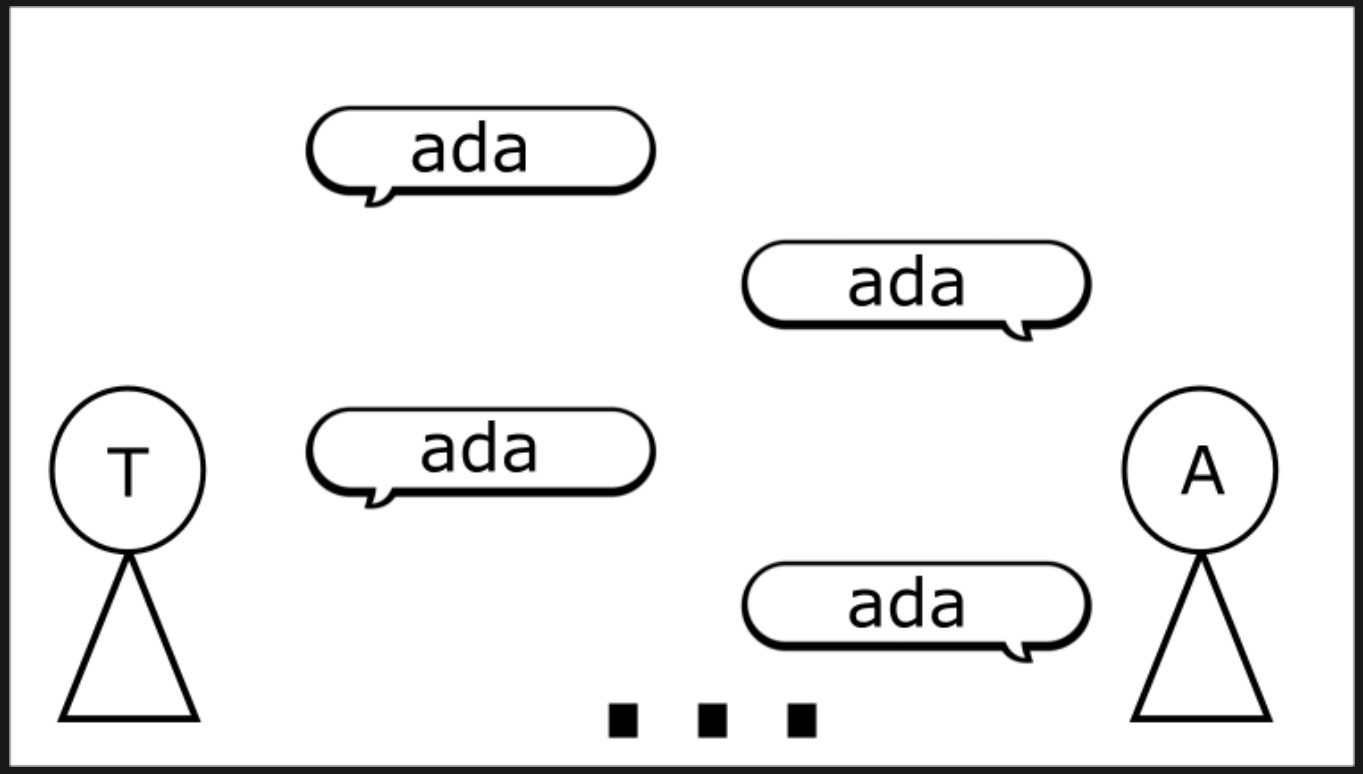
青木君の勝ち

bcda



高橋君の勝ち

ada



引き分け

こんな感じでどちらが勝つかを判定する問題です
今回の例であれば簡単に求めることができますが、制約を見ると

N は $2*10^5$ 以下の整数

となっております。ざっと、20万単語あります。
これを全部試していくのはなかなかに大変ですね。

どちらが勝つかを効率よく求めたいのですが、

ここで問題となるのはこの一言です。

ただし、二人とも最善に行動するとします。
具体的には、自分が負けないことを最優先し、
その次に相手を負かすことを優先します。

自分が負けないことってどうするの？

どうすれば相手を負かす選択になるの？

アルゴリズム説明

最善に行動する
ということについて説明をしていきます。
一見難しそうですが、当たり前じゃん、という小さな気づきを積み上げていきます。

説明の都合上
２つほど正しくないことを言いながら進めます
後程、訂正しますので何卒ご容赦ください

先ほどの問題は一旦置いときます

いきなりですが、簡単なゲームをしましょう

あなたは次の３つから好きな手を選べます

- この手を選ぶとあなたは勝ちます
- この手を選ぶとあなたは負けます
- この手を選ぶと引き分けになります

どの手を選びますか？

この場合には

この手を選ぶとあなたは勝ちます

を選ぶと無事に勝利することができます。

とっても簡単ですね。

続いて、、この場合はどうでしょうか

- この手を選ぶとあなたは負けます
- この手を選ぶとあなたは負けます
- この手を選ぶと引き分けになります

この場合には

この手を選ぶと引き分けになります

を選ぶかと思います。どうしても勝てないのはしょうがないですが、負けは嫌ですもんね。
苦肉の策とでもいいましょうか。

最後です、もう１つだけお付き合いください

- この手を選ぶとあなたは負けます
- この手を選ぶとあなたは負けます
- この手を選ぶとあなたは負けます

最悪な状況ですね。この場合には

この手を選ぶとあなたは負けます

を"選ぶしかない"という表現が適切でしょうか。
悔しいですが何をやっても勝てません。敗北を受け入れましょう。

まとめてみます

- 勝ちが1つでもある：勝ち
- 勝ちが無く引き分けがある：引き分け
- 負けしかない：負け

優先したい手が１つでもあればそれを選ぶと考えると分かりやすいかもしれないです。

ここまで理解できたでしょうか？
この時点で後退解析の８割は理解できたといっても過言ではありません。 蓋を開けてみればとても単純です。

説明してきましたが、よくよく考えると少しだけおかしいです

ここで訂正１です

実際の問題では高橋君と青木君の２人で行われているため

毎回ターンが入れ替わります。

そのため勝ち負けの条件が少し変わります

それらを含めて勝ち、負けを定義すれば正しいと言えなくもないのですが、、

条件を考えるポイントは

相手にその選択肢を押し付ける

ということです。

少しだけ言い回しを変えます

この場合にはどれを選べばいいでしょうか

- この状態になると勝ちです
- この状態になると負けです
- この状態になると引き分けです

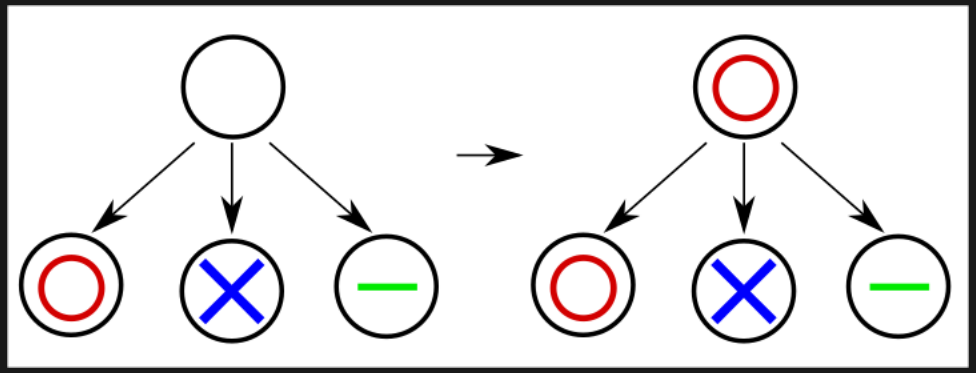
ポイントは選んだ手番は相手になるということです。

この場合には

この状態になると負けです

を選ぶと勝てますね。

負けが決定している手を押し付けちゃいました。



もう一つ考えてみます

今度はどうでしょうか？

- この状態になると勝ちです
- この状態になると勝ちです
- この状態になると勝ちです

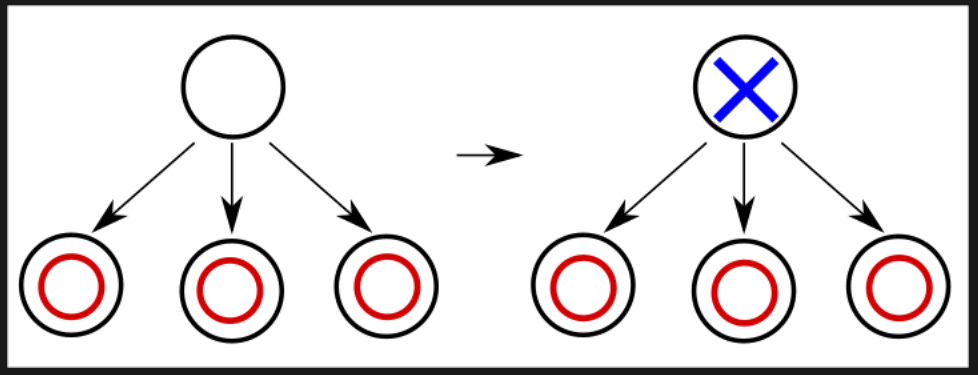
この場合は選択肢はありませんね

この状態になると勝ちです

しかありません。

次のターンは相手の番ですので、勝利するのは相手です。

残念なら負けてしまいました。



では「引き分け」はどうなるのでしょうか？

今までは

勝ちが無く引き分けが１つ以上ある

と説明しました。が、、

ここで訂正２です

「引き分け」は
「勝ち」でも「負け」でもない
状態です。少しばかり漠然としていますが、
納得していただけるでしょうか？

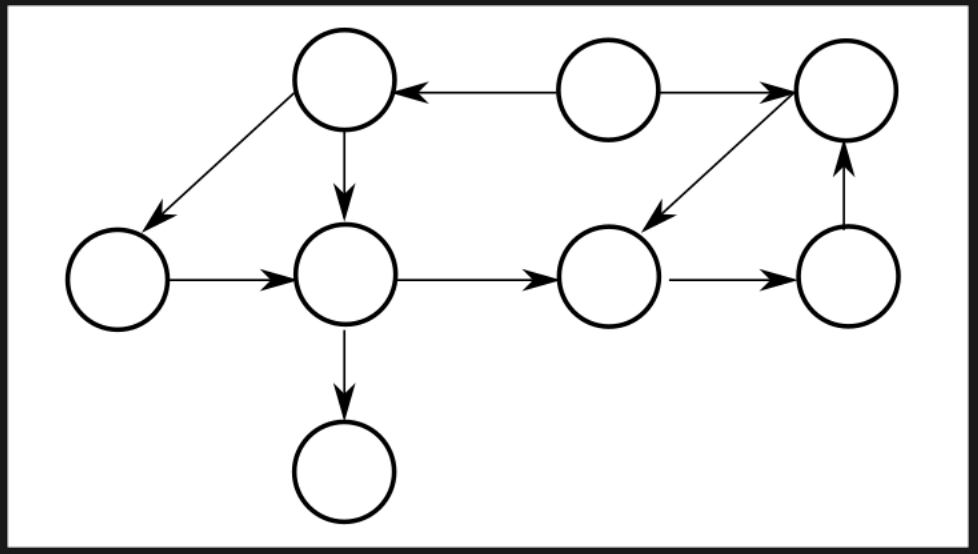
再度まとめてみます

- １つでも負けの状態を選択可能：勝ち
- 勝ちの状態のみ選択可能：負け
- 勝ち、負けのどちらでもない：引き分け

以上の条件が正しい条件です。以降はこれを元に考えていきます。

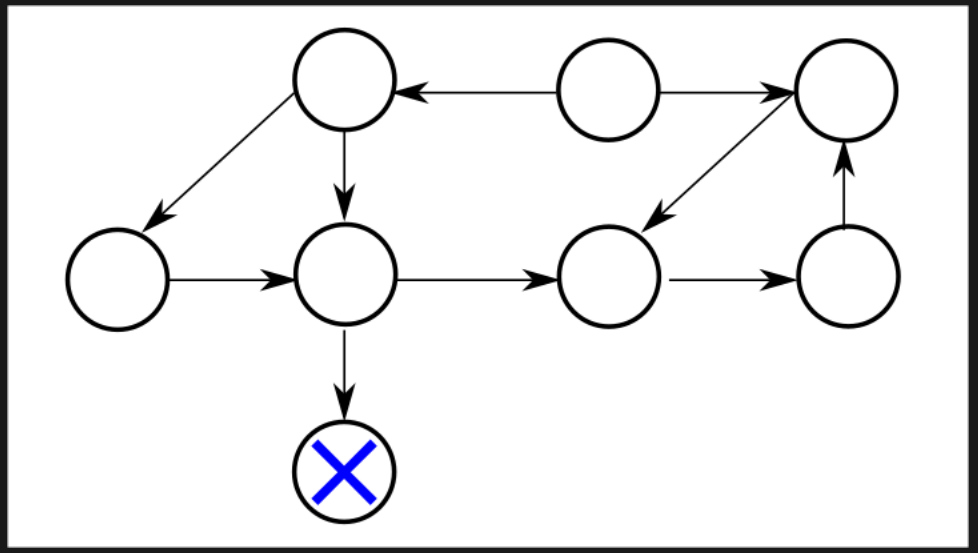
以下の遷移をするゲームを考えてみます

いきなり、
ゲーム？グラフ？
となる方がいらっしゃるかもしれないですが、
問題をグラフにするのは後程にしますので「こういうものがあるんだな」で大丈夫です。



まず終点の状態は負けになります。

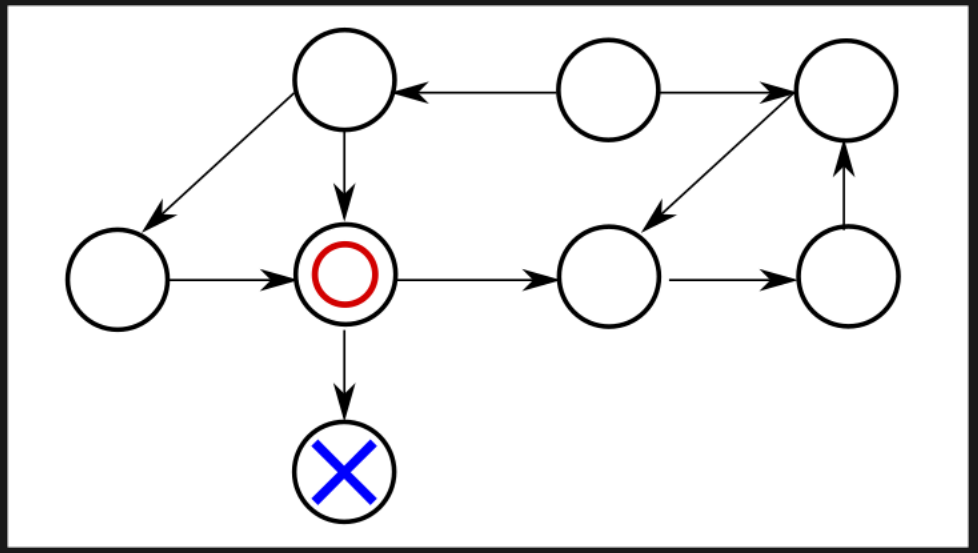
これ以上先はないのでしょうがないですね



次に終点に向かう頂点を考えます。

終点は**負け**が決まってますので、

終点に向かう頂点は**勝ち**となることが分かります。

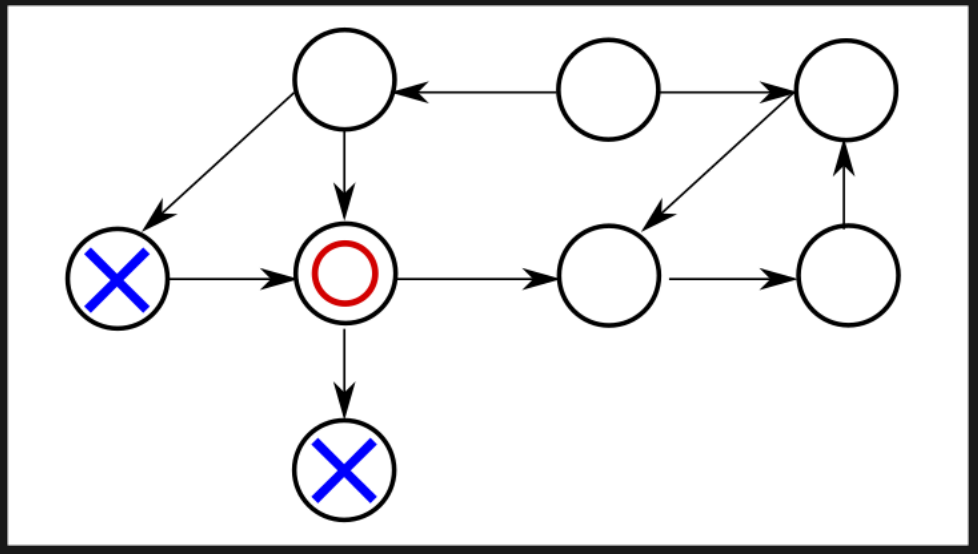


もう一つだけ見ていきます

該当の頂点は先ほどの **勝ち** の頂点にしか遷移できません。

つまり相手に **勝ち** の手番を渡すしか選択できません。

残念ながらこの頂点は **負け** になってしまいました。

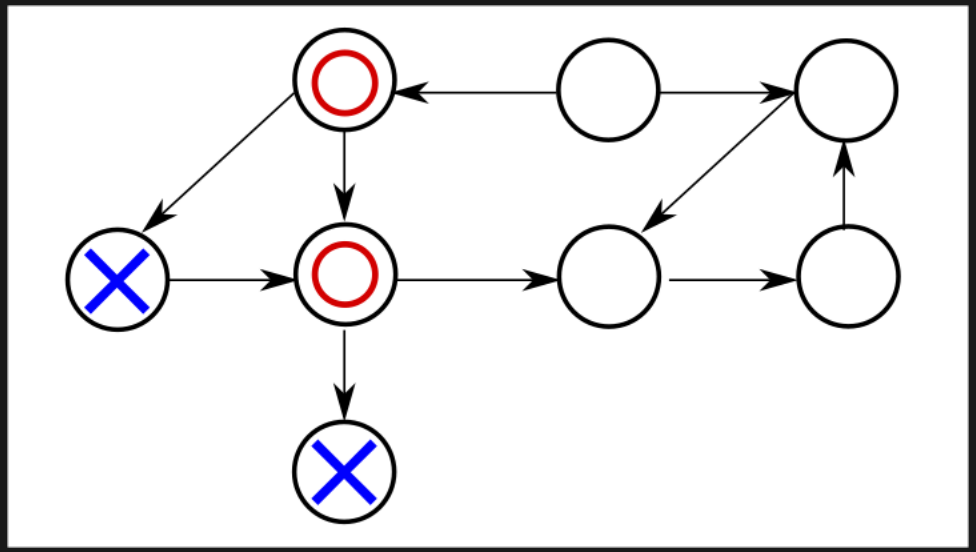


また現段階で判断できない場合には後回しにしておきましょう。
他の頂点を決定していくと
いつの間にか決まっているかもしれません。

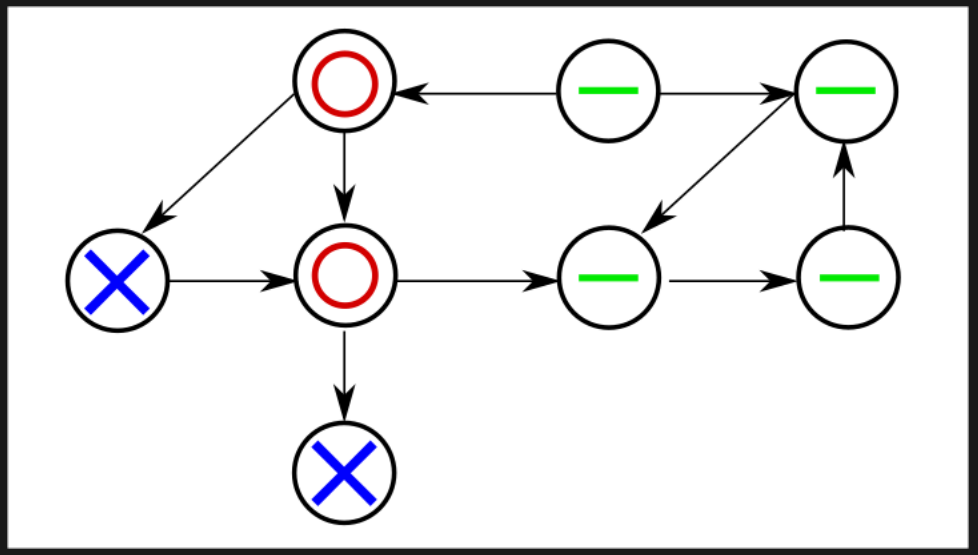
具体的には、、

負け には遷移できないけど、まだ選択の余地がある場合

こんな感じで最後まで決めていきます



最後まで決まらなかった頂点は
引き分け ということにして探索終了です。



探索順について

グラフの説明で探索順について話しました

後退解析の探索順は以下の通りです

- 選択の余地が無くなってしまった頂点:負け
- 負けの頂点に移動できる頂点：勝ち

各状態の
「勝ち」「負け」「引き分け」
を求めることができました。

以上が後退解析の説明となります。

問題解説

後退解析を使って先ほどの問題を実際に解いていきます。

ルールに従って単語をつなげます

高橋君が

abcd

と言った場合、青木君は

bcd

から始まる単語を言わなければなりません

青木君は bcd から始まる単語

bcda

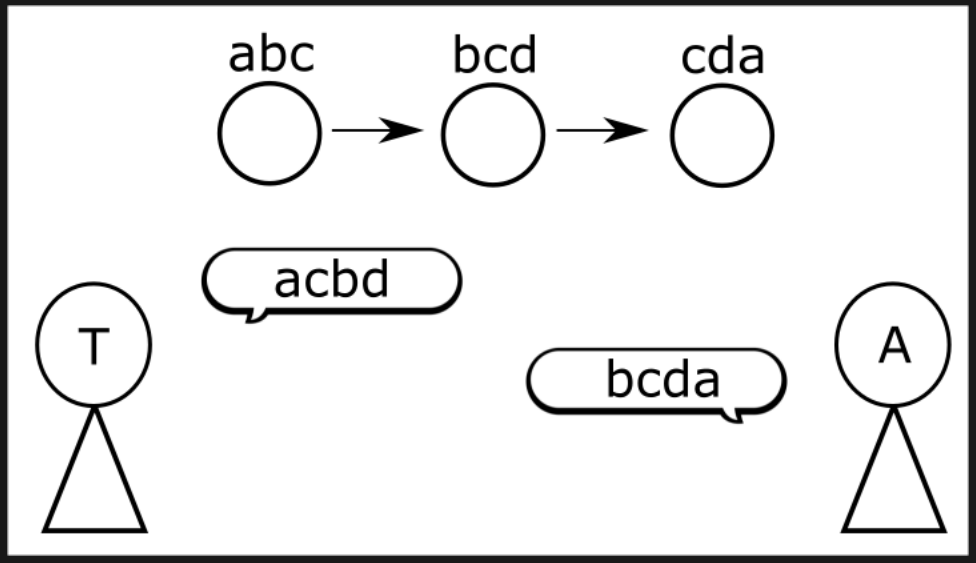
を言いますので、次に高橋君は

cda

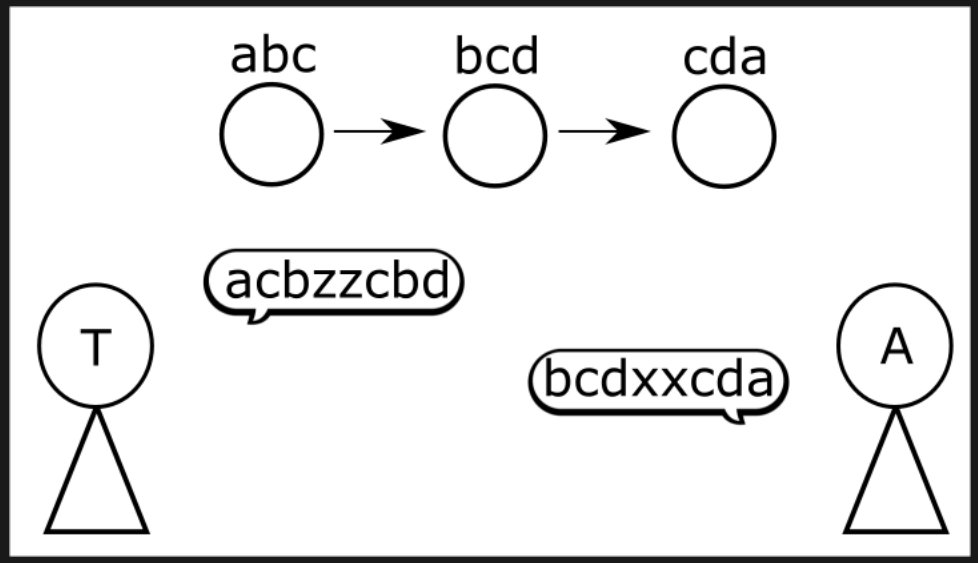
から始まる単語をいいます

が、そんな単語はありませんので高橋君の負けとなります

先ほどの流れは以下のようになります



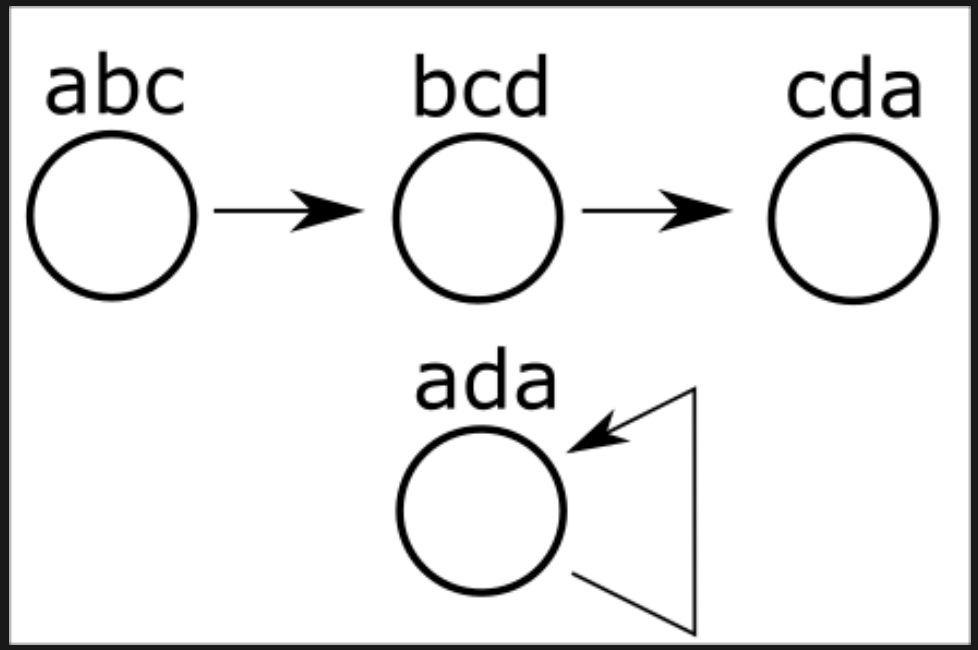
先ほどと同じ状態が作れるのであれば
単語は何でも大丈夫です



- 同じ単語は何回でも使ってよい

そのため各単語の前後3文字だけ分かればよいです

入力例のグラフ



先ほどと同様に後退解析によって勝敗を考えていきましょう

実装のポイント

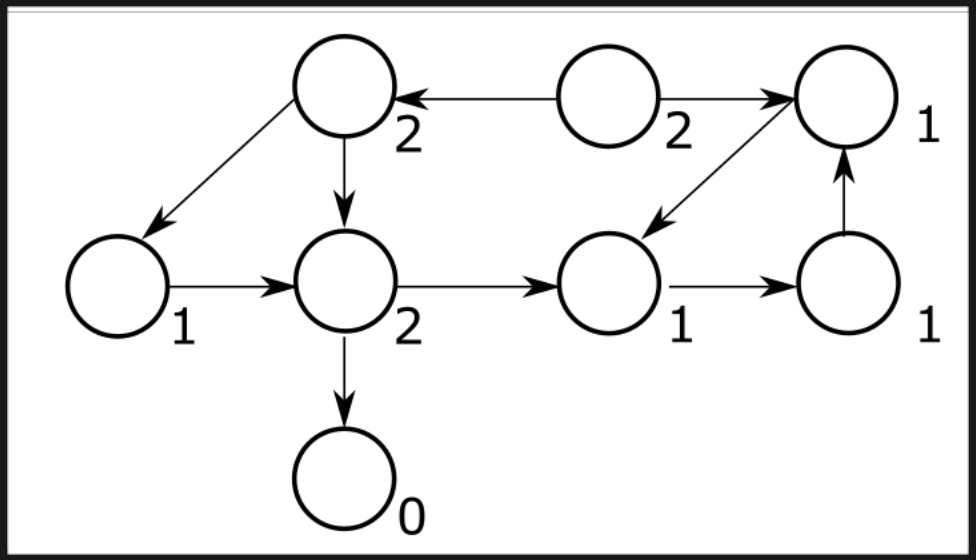
各頂点の持つ選択肢の数

に注目してあげるといい感じに実装できます。

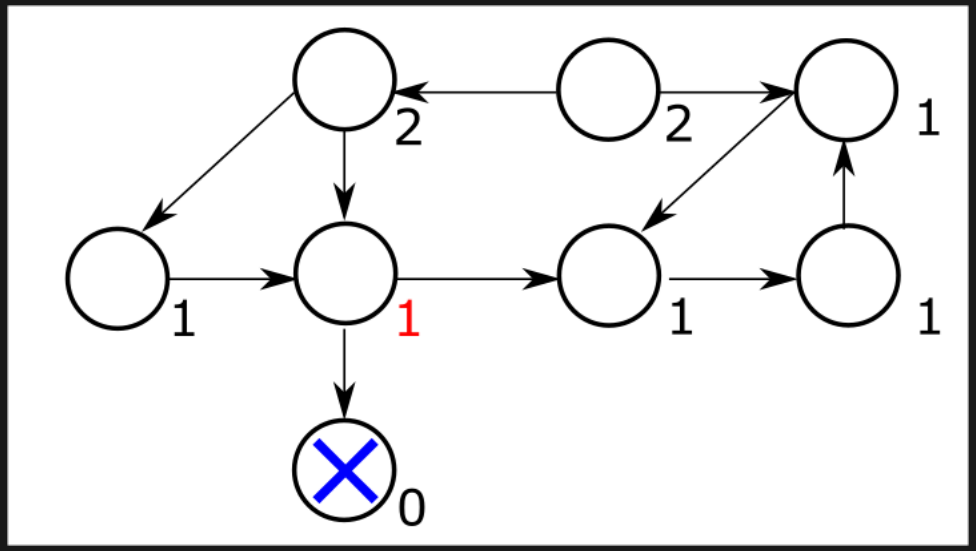
判断ができない場合には後回しにしてましたよね

そのため選択肢の数（出次数）が 0 の頂点に注目するわけです

選択肢の数（出次数）に注目したグラフ

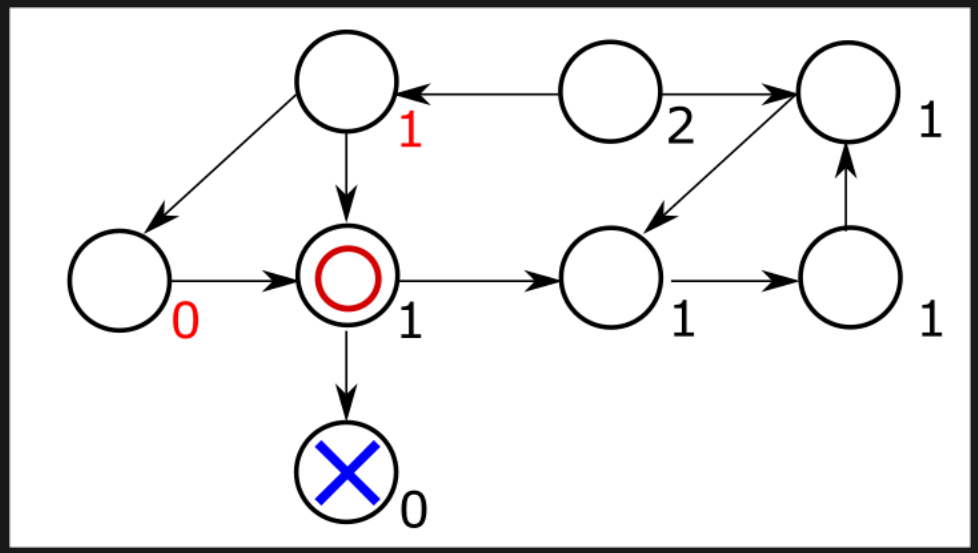


終点＝出次数が 0 は負けです



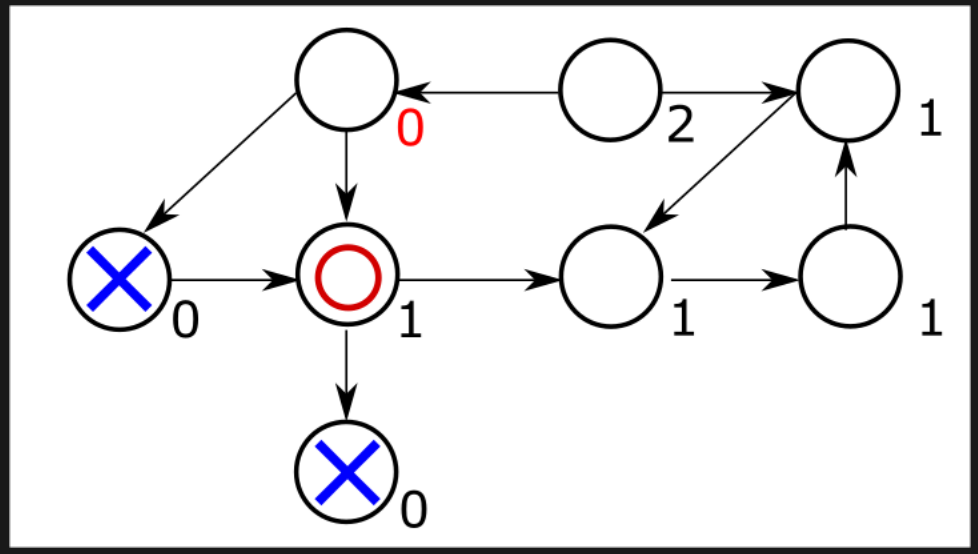
このときに繋がっている頂点の出次数を1つ減らします

負けに遷移できる頂点は勝ちです



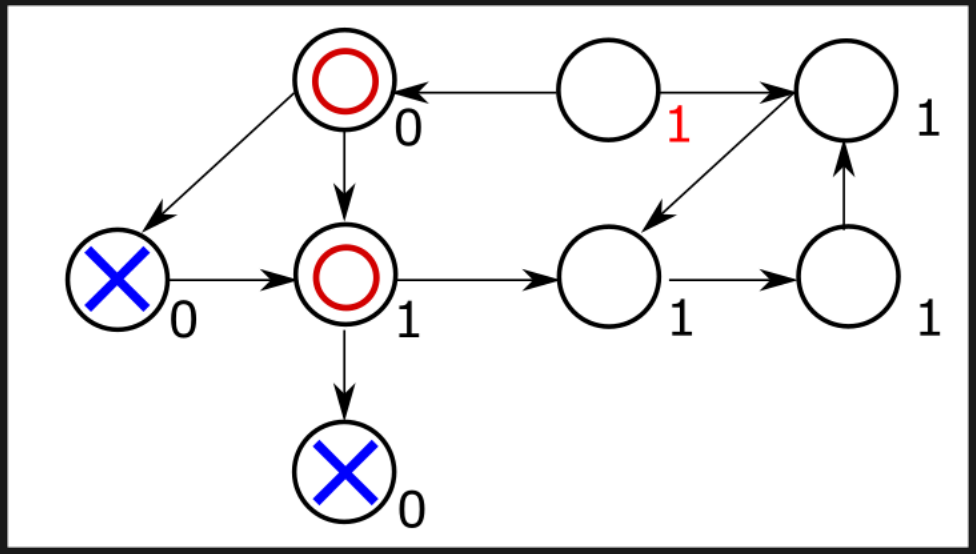
ここ出次数関係ないです。すみません

出次数が 0 の頂点が誕生しました
これを負けとします



このとき繋がっている頂点の出次数を1つ減らします

操作を行えなくなるまでこれを繰り返します



実装

- C++
- python

計算量

実装のポイントでお話しましたが本問題は

選択肢の数（出次数）

を減らすように更新を行いました。

更新が行われるのは辺で繋がっている箇所ですね。

しかも更新は１つの辺につき高々１回となります。

また、辺の数は単語数 $N \leq 2*10^5$ で抑えることができます

（各単語の前３文字と後ろ３文字に辺を貼りました）

頂点数にも注目しておきます

本問題では単語の前後３文字に注目しました

そのため、取りえる状況は実は余り多くありません。

文字の種類数：英小文字と英大文字　52種

文字数：3文字

52*52*52=140608

です。$1.5*10^5$ ぐらいですね

おわりに

後退解析

のアルゴリズムを使って問題を解いてみました。

どうでしたでしょうか？我々の頭を悩ませる

最善な行動を取る

の気持ちは少しは理解できたでしょうか？

一見難しそうな内容でも、その内容を覗いてみると
簡単、単純、当たり前
の組み合わせだったりします。
アルゴリズムのとっても面白いところですね

参考

アルゴリズム実技検定 公式テキスト[エントリー~中級編]

（amazonのリンクです！購入してもぼくには一銭も入らないのでご安心ください）

PAST本と呼ばれているやつです。グラフの説明はこちらを参考にしています。

サンプルコードはpythonですので、C++erはむしろ良い練習になるかもです。

ゲームを解く！Educational DP Contest K, L 問題の解説

けんちょさんの記事です。類題はこちらに載っております

いつも大変お世話になっております

長時間のご清聴ありがとうございました