JOI 2015-2016 本選 問題 3
鉄道運賃 (Train Fare) 解説

三谷庸

問題概要

* N 個の駅、M 個の鉄道路線がある
* Q 年間にわたる値上げ計画がある
* 値上げ計画の i 年目には、路線 R_i の運賃を値上げする
* etc

* グラフ理論の言葉で言い換えます

問題概要

* N 頂点 M 辺の無向グラフがある
* はじめ、すべての辺のコストは 1 である
* クエリが Q 回くる
* i 番目のクエリでは辺 R_i のコストを 2 にする
* 各クエリに対して、その時点で以下の条件を満たす頂点 v の個数を出力する
  * 頂点 v から頂点 1 への最短距離がはじめの状態よりも長くなっている

小課題 1

* N <= 100
* Q <= 30

* とにかく問題文に書いてある通りにやる
* 各頂点 v について、「頂点 v から頂点 1 への最短距離」を求めたい

小課題 1

* 最短距離を求めるには、Warshall-Floyd 法を用いればよい
* N 頂点のグラフでは、全点対最短距離が O(N^3) で求まる
* クエリごとに全点対最短距離を求める
* 全体の計算量は O(QN^3) となる

考察: 始点をひとつに

* 求めたい最短距離はN-1種類あり、始点は全て異なるが、終点は共通
* これを生かして速くできないか

考察: 始点をひとつに

* 求めたい最短距離は N - 1 種類あり、始点は全て異なるが、終点は共通
* これを生かして速くできないか

* 無向グラフでは、s-t 最短距離と t-s 最短距離は等しい
  * s-t 最短路を逆からたどると t-s 最短路になる

考察: 始点をひとつに

* したがって、今回の問題は次のように言い換えられる
  * 頂点 1 から頂点 v への最短距離が初めの状態より長くなっている v の個数を求めよ

考察: 始点をひとつに

* したがって、今回の問題は次のように言い換えられる
  * 頂点 1 から頂点 v への最短距離が初めの状態より長くなっている v の個数を求めよ
* 頂点 1 からほかのすべての頂点への最短距離を求めるには、Dijkstra 法を用いればよい

小課題 2

- N <= 100,000
- M <= 200,000
- Q <= 30
- クエリごとに、頂点 1 を始点として Dijkstra 法を行う
- 前ページのように問題を言い換えて、条件を満たす v を数える
- 全部で O(Q M log N) 時間

考察: コスト 2 の辺

* 頂点 1 から頂点 v への経路で、以下の条件を満たすものを「よい経路」と呼ぶことにする
  * コストが初めの状態での最短距離と等しい
* よい経路にはコスト 2 の辺は現れない

考察: コスト 2 の辺

* 頂点 1 から頂点 v への経路で、以下の条件を満たすものを「よい経路」と呼ぶことにする
  * コストが初めの状態での最短距離と等しい
* よい経路にはコスト 2 の辺は現れない
  * もし現れていたとすると、初めの状態では、全く同じように辺をたどってより小さいコストで頂点 1 から v へ移動できる

小課題 2 (別解)

- 各クエリにおいて、指定された辺のコストを 2 にするかわりに、その辺を削除する
- この場合、(残っている)すべての辺のコストが 1 となるので、最短距離を求めるのに幅優先探索 (BFS) を用いることができる
- 計算量は O(QM)

小課題 3

* N <= 100,000
* Q <= M <= 200,000
* **「正しい出力にあらわれる整数は 50 種類以下」**

小課題3

- N <= 100,000
- Q <= M <= 200,000
- 「正しい出力にあらわれる整数は50種類以下」

- クエリごとにグラフ全体を見ていては間に合わない
- 最後の条件をどう使うか
- 「答えが変わった」ことを検出したい

考察: 最短路に現れる辺

- 前の考察で、コスト 2 になった辺は考えなくてよいことが分かった
- 他の辺は全て考慮すべきだろうか

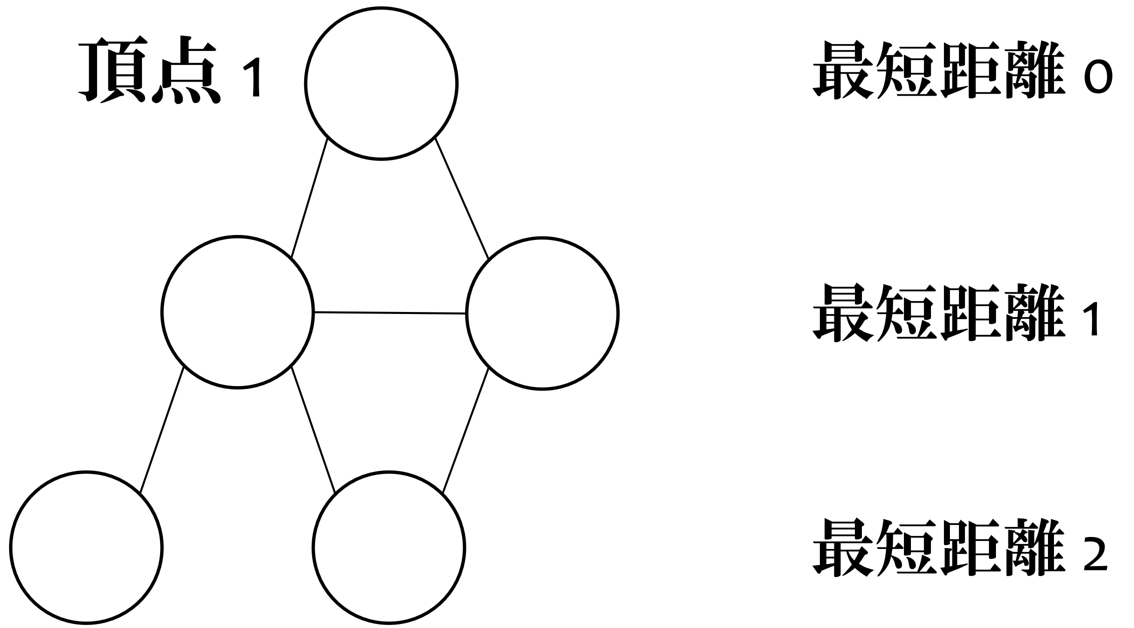
考察: 最短路に現れる辺

- 初めの状態のグラフで、各辺が頂点 1 からの最短路でどのように使われるか考える

- N 個の頂点を頂点 1 からの最短距離で分類する
  - 辺はどのような頂点を結んでいるだろうか？
  - それらの辺は最短経路でどのように使われるだろうか？

考察: 最短路に現れる辺

* 入力例 1 のグラフでやってみる



考察: 最短路に現れる辺

* 以下のことがわかる
  * 辺の両端点の頂点の頂点 1 からの最短距離は (d, d) か (d, d + 1) のいずれか
  * (d, d) の辺はどの最短路にも使われない
  * (d, d + 1) の辺は d -> d + 1 の方向でのみ最短路に使われる

考察: 最短路に現れる辺

- 前述の観察の証明(3 つ順に)
  - 例えば (d, d + 2) の辺 (u, v) があれば、u まで距離 d で行った後この辺を使うことで v まで距離 d + 1 で行けてしまう
  - (d, d) の辺 (u, v) が最短路に u -> v の向きで使われたとすると、v までは距離 d、したがって u までは距離 d - 1 で行けてしまう
  - 3 つめも同じように考えると分かります

考察: 最短路に現れる辺

* いくつかの辺のコストが 2 になっても、よい経路 (最短距離が初めと同じに保たれている最短路) で使われる辺は前の形だけ
* 初めのグラフで 1 回だけ BFS することで、最短路に使われる辺を列挙できる
* どの最短路にも現れない辺を削除するクエリは単に無視すればよい

考察: 最短路に現れる辺

* 最短路に現れる辺以外は無視する
* また、残った辺も最短路に使われる向きに限定する

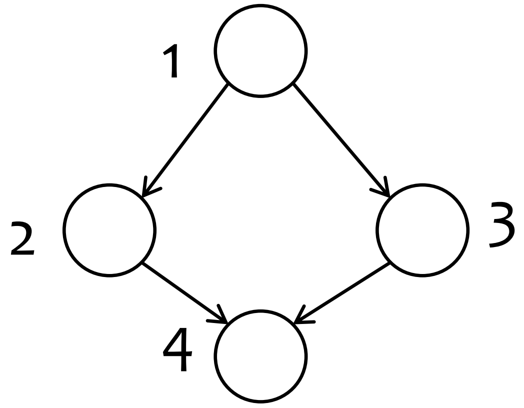
* このグラフは DAG (Directed Acyclic Graph, 閉路がない有向グラフ) となる

考察: 最短路に現れる辺

- このグラフの辺を消していくと考える
- 頂点 1 から頂点 v へのよい経路 (コストが初めの状態での最短距離と等しい経路) が残っていることと、このグラフ上で頂点 1 から頂点 v への経路が残っていることは同値

考察: 最短路に現れる辺

* この DAG の辺を消すことと、クエリに対する答えが変わることは同値ではない！！
* 下のグラフで辺 (3, 4) (右下の辺)を消しても経路が無くなる頂点は存在しない



考察: 最短路が消えるとき

* この DAG 上で辺を消してゆく
* 辺 u -> v を消すと、頂点 1 から頂点 v への経路がなくなったとする
* これはどのような状況か？

考察: 最短路が消えるとき

* この DAG 上で辺を消してゆく
* 辺 u -> v を消すと、頂点 1 から頂点 v への経路がなくなったとする
* これはどのような状況か？

考察: 最短路が消えるとき

* この DAG 上で辺を消してゆく
* 辺 u -> v を消すと、頂点 1 から頂点 v への経路がなくなったとする
* これはどのような状況か？

* 「この辺を消すことで頂点 v に (直接) 入る辺がなくなった」ということである

考察: 最短路が消えるとき

* したがって、各頂点について入次数 (その頂点に向かう辺の数) を記録しておくと、各クエリに対して「その辺を消すことでよい経路が無くなる頂点 v が存在するか」が O(1) で判定できる

考察: 最短路が消えるとき

* 辺 u -> v を消しても頂点 v へのよい経路が無くならない場合、すべての頂点についてよい経路があるかどうかは変わらない
* 頂点 v へのよい経路が無くなった場合、v 以外にもよい経路が無くなる頂点があるかもしれない
  * 例えば、1 -> 2, 2 -> 3 の 2 辺のみのグラフで辺 1 -> 2 を消すと、頂点 3 への経路もなくなる

小課題 3

* 「正しい出力に現れる整数は 50 種類以下」という条件が使えるようになった

* 辺を消すごとによい経路が無くなる頂点があるか判定し、ある場合はグラフ全体について BFS して各頂点についてよい経路の存在を判定する

小課題 3

* もう少し詳しく説明

init_dis[v] := 元のグラフでの頂点 1 から頂点 v への最短距離

cur_dis[v] := その時点での最短距離

として、

* 元のグラフの辺 (u, v) が u -> v として DAG に含まれることは、init_dis[v] = init_dis[u] + 1 と同値

小課題 3

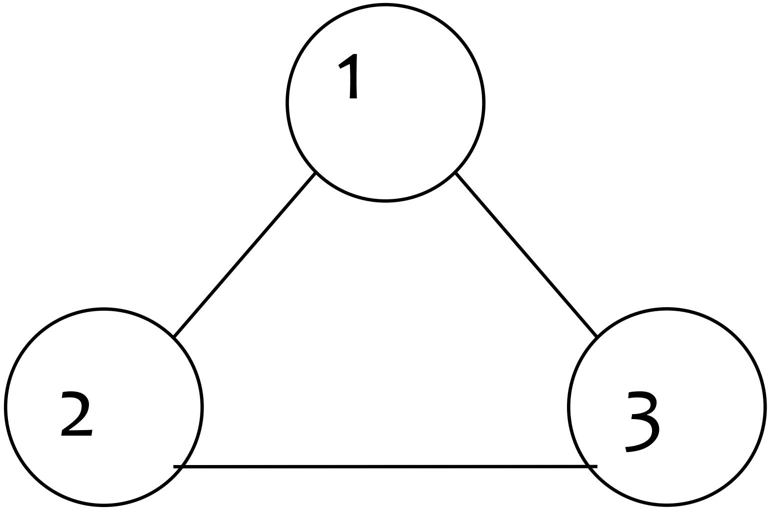
- 辺を消してよい経路の存在性が変わった時は、新たなグラフで BFS して cur_dis[v] を求める
  - DAG は明示的には持たずに、元のグラフからクエリの通りに辺を消していくだけでよい
- その後、cur_dis[v] == init_dis[v] となる v の個数を数える

小課題 3

* 新たなグラフで u -> v がよい経路に含まれることと、cur_dis[v] == cur_dis[u] + 1 は同値ではない！！
* 次のページのグラフで辺 (1, 3) を消すと、cur_dis[3] = cur_dis[2] + 1 だが、辺 (2, 3) は DAG の辺として考えてはいけない

小課題 3

* (グラフの例)



小課題 3

* 各頂点に対して入次数 indeg[v] を持っておく
* u -> v を消したときに indeg[v] -= 1 すべきかどうかの条件は、次(の両方を満たすこと)が正解
  * cur_dis[v] == init_dis[v]
  * cur_dis[v] == cur_dis[u] + 1

小課題3

* まとめると、辺 (u, v) を消すクエリでは以下のことをすればよい

よい経路の存在性が変わるかどうかを indeg から判定

変わるときは、(O(M) かけて) すべての頂点について最短距離を計算しなおす

小課題 3

* 計算量は O(Q + TM) (= O(TM)) (Q <= M なので)
  * ただし T は出力に現れる整数の種類数

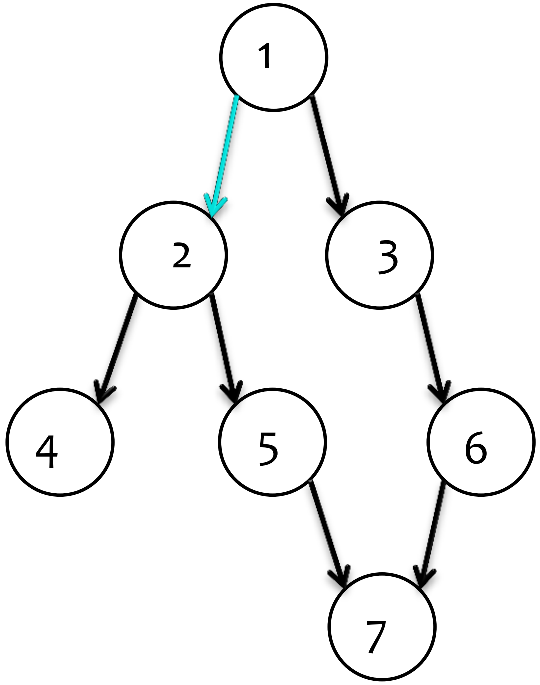
小課題 4

- Q <= M <= 200,000

- 同じ辺を何度もみていては間に合わない
- 小課題 3 の T 回の BFS では似たような計算をやっていそう
- 各頂点について「その頂点への経路が消えるのはいつか」を求めたい

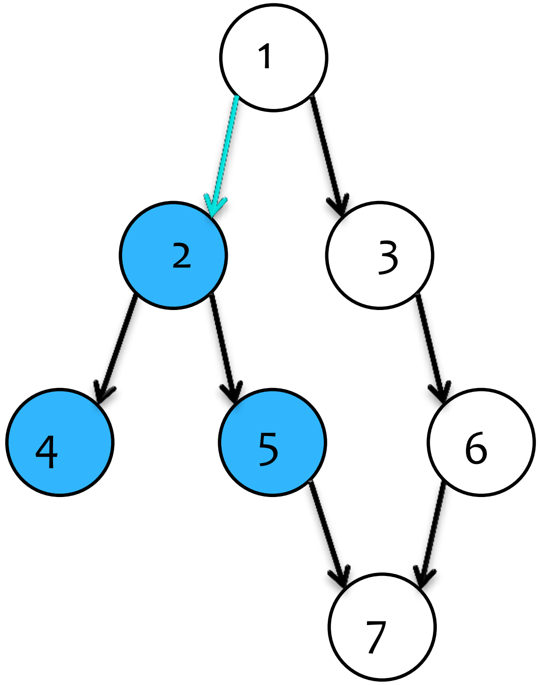
経路がなくなる頂点の列挙

* 下のグラフで、辺 1 -> 2 (左上の水色の辺) を削除するとどうなるか



経路がなくなる頂点の列挙

* 水色に塗られた頂点への経路がなくなる



経路がなくなる頂点の列挙

* 頂点 2 からいくつかたどっていける頂点が水色に塗られている

経路がなくなる頂点の列挙

* 頂点 2 からいくつかたどっていける頂点が水色に塗られている
* これらの頂点は、頂点 2 から DAG 上で BFS (DFS でもよい) すれば列挙できる
* BFS を使って経路がなくなる頂点だけを記録したい (前のグラフの場合、頂点 7 (一番下の頂点) は記録しないようにしたい)

経路がなくなる頂点の列挙

- BFS で頂点 v を選んでいるとき、次の条件が満たされるならば、v には経路がなくなったと記録せず、v から出る辺は見ない
- 逆に、満たされないならば、v を記録して、BFS を継続する
  - BFS で見ていない辺だけを使う頂点 1 から頂点 v への経路が存在する

経路がなくなる頂点の列挙

* 条件を満たすかどうかは次のようにして判定する

indeg[v] := (頂点 v へ向かう辺で、よい経路に含まれうるものの個数) となるようにする

BFS で u を選んでいるとき、辺 u -> v の処理は、

```
indeg[v] -= 1
if indeg[v] == 0
	v を経路がなくなった頂点として記録
	BFS を継続
```

小課題 4

* 以上のようにして、辺を削除した時にそれによって経路がなくなる頂点を列挙できる
  * DFS を使う場合もほぼ同じことをすればよい
* 経路がない頂点の個数をあらわす変数を持っておき、頂点への経路がなくなるたびに増やせばよい
* 計算量は？

小課題 4

* 最大 Q 回にわたる BFS (または DFS) において、辺 u -> v を見るのは、頂点 u が初めて到達不可能になった (経路がなくなった) 時の 1 回だけ
* よって、計算量は Q 回のクエリすべて合わせて O(M)

小課題 4

- BFS の初期条件や終了条件を間違えると TLE する可能性があります
- そうなったときは、「同じ辺を 2 回以上みない」アルゴリズムになっているかよく確認してみましょう

小課題４別解

* 最短路に関する考察はこれで終わりですが、少し違った方針で実装することもできます

小課題４別解

* 最短路に関する考察はこれで終わりですが、少し違った方針で実装することもできます
* 辺を「削除」するよりも「追加」する方がやりやすそう

小課題４別解

* クエリを逆から見る

* どのクエリにも現れない辺をあらかじめ追加しておく
* その後、後ろからｉ個のクエリの辺を追加した状況は、前からＱ-ｉ個のクエリの辺を削除した状況と同じになる

小課題 4 別解

* クエリの辺を逆から追加していき、各頂点についてどの時点で頂点 1 から到達可能になったかを調べることにする

小課題 4 別解

- 到達可能 (頂点 1 からの経路がある) な頂点 u と到達不可能な頂点 v について辺 u -> v を追加するとき、頂点 v は到達可能になる
- 頂点 v 以外にも到達可能になる頂点があるかもしれない
- そのような頂点は BFS (DFS でも)で列挙できる
  - 前の解法で探索を打ち切る条件を「到達不可能な頂点」から「到達可能な頂点」に変えるだけ

小課題 4 別解

* BFS の終了条件などは以下の通り

辺 u -> v の処理は

if (v への経路が存在)

v はこの BFS では無視 (すでに以前の BFS で記録済み)

else

que.push(v)

v への経路が存在すると記録

BFS を継続

小課題 4 別解

* 計算量は同じく O(M)
  * 辺 u -> v は頂点 v が初めて到達可能になるときだけみられる

小課題 4 別解

* (今回は逆から見なくても解けましたが) クエリを逆から見るという手法はとても重要です

小課題 4 別解 2

* 各辺について、その辺が削除されたのは何番目かを持っておく
* Dijkstra 法 (更新の方法を工夫する) を 1 回だけ行うことで、各頂点 v について以下の値が分かる
  * 頂点 1 から頂点 v まで、i 番目のクエリまでは残っている辺だけで最短距離でたどり着けるような最大の i
  * 最短距離の代わりに (距離, 経路が消える時間) を求めるようにすればよい
* これを使うとクエリに答えられる
* 計算量は O(M log N)

余談: 最短路問題

* プログラミングコンテストでは、最短路アルゴリズムとしては以下を覚えておくとよいでしょう
* 重みなしグラフの場合
  * 幅優先探索 (BFS)

余談: 最短路問題

* 重みつきグラフの場合
  * Warshall-Floyd 法
  * Bellman-Ford 法
  * Dijkstra 法
* 各アルゴリズムについて、計算量、適用できる条件を確認しておきましょう
  * なお、Dijkstra 法はコスト 0 の辺があっても使えます

得点分布