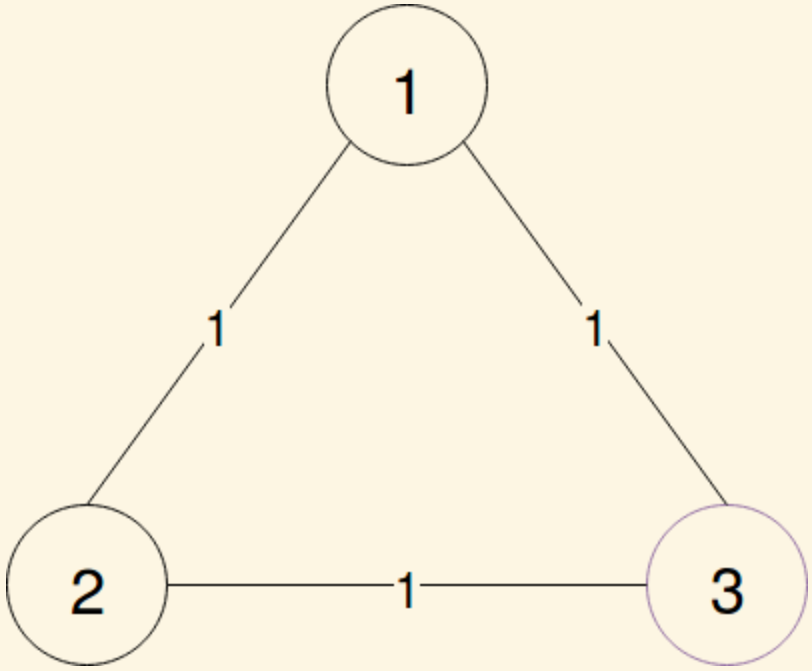
イノシシ (Wild Boar)

joisino

課題の説明

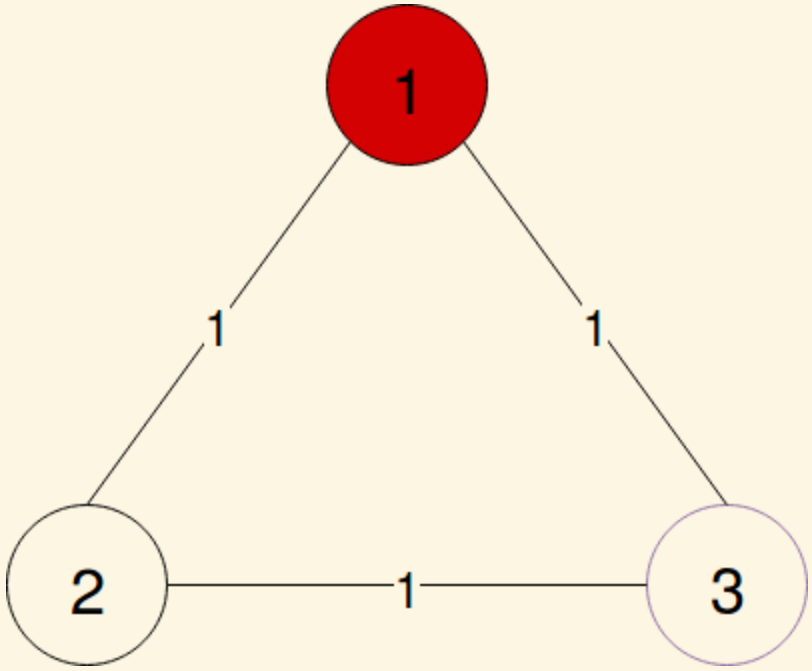
- 重み付き無向グラフと頂点列が与えられる
- 頂点列をこの順に訪れる最小合計コストを求める
- ただし、辺を通ったあとすぐにその辺を折り返すことはできない
- 頂点列の一つを変更するクエリが与えられるので、変更するたびに最小合計コストを求める

サンプル 1



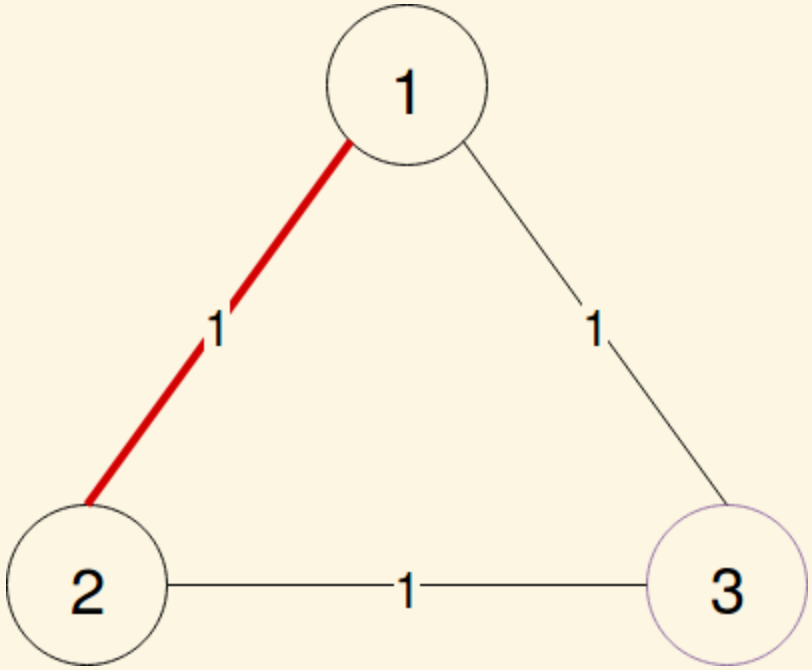
- 実行する補給計画: 1 -- 2 -- 1
- 時間: 0

サンプル 1



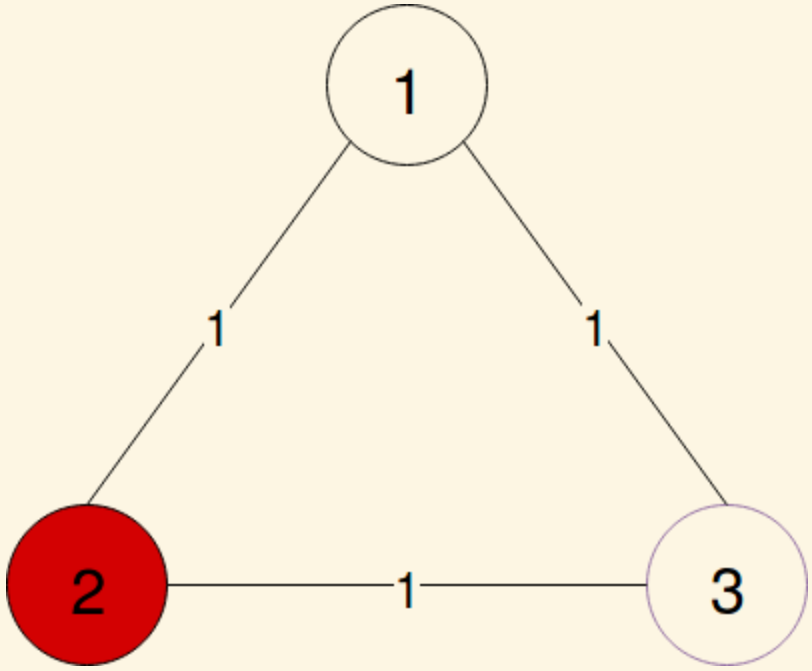
- 実行する補給計画: **1** -- 2 -- 1
- 時間: 0

サンプル 1



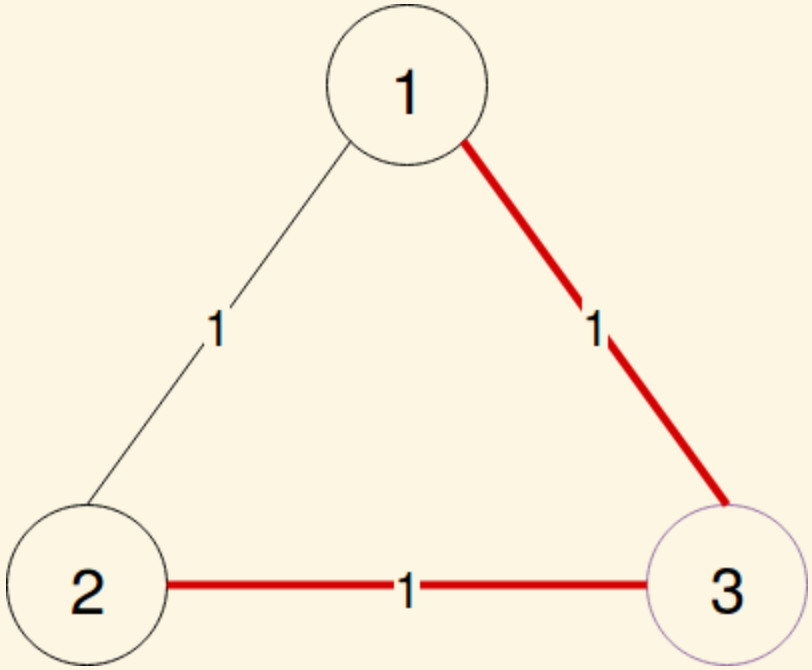
- 実行する補給計画: 1 -- 2 -- 1
- 時間: 1

サンプル 1



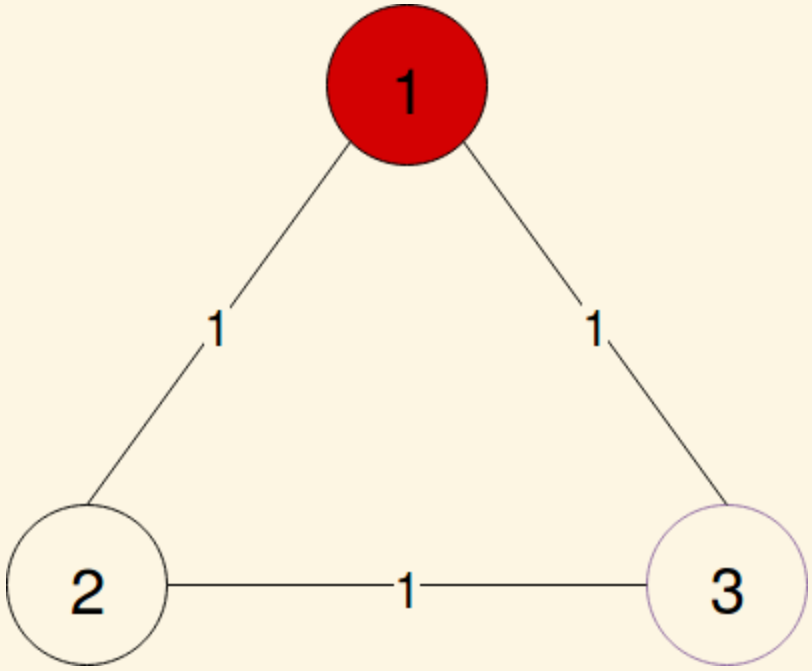
- 実行する補給計画: 1 -- **2** -- 1
- 時間: 1

サンプル 1



- 実行する補給計画: 1 -- 2 -- 1
- 時間: 3

サンプル 1



- 実行する補給計画: 1 -- 2 -- **1**
- 時間: 3

サンプル 1

- 先に損する逆回りでも OK

小課題 1 (12 点)

- `N <= 10`
- `M <= 10`
- `T = 1`
- `L <= 10`
- `C_i <= 10`

全てがとても小さい。

解法

- 全てがとても小さい
- 例えば(何番目の補給地点まで訪れたか, 現在の頂点, 直前の頂点) を状態として dijkstra すれば解ける
  - 頂点数: `O(LN^2)`
  - 辺数: `O(LNM)`
  - 時間計算量 `O(LNM log (LN^2))` など
- これ以外にも最短路問題 or DP に帰着できれば大抵 OK

小課題 2 (35 点, 合計 47 点)

- `N <= 500`
- `M <= 500`
- `T = 1`

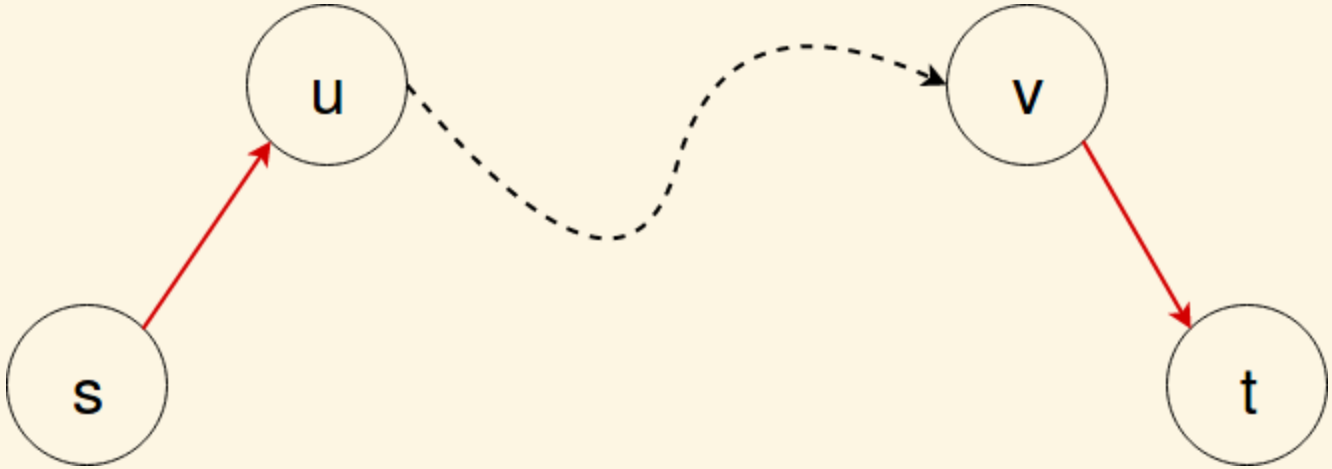
頂点数と辺数は小さい。クエリは 1 回。

考察

- 補給地点は多い (`L <= 100,000`) -> 補給地点の数にできるだけ依らないようにしたい
- 補給地点間のパスは毎回計算しなくてもあらかじめやっておけばよさそう

考察

- 全ての頂点組 `(s, t)` と `(s, t)` に接続する辺 `(s -- u, v --t)` について, 最初に `s -> u` と辺を使って最後に `v -> t` と辺を使う時の最短路長を計算する

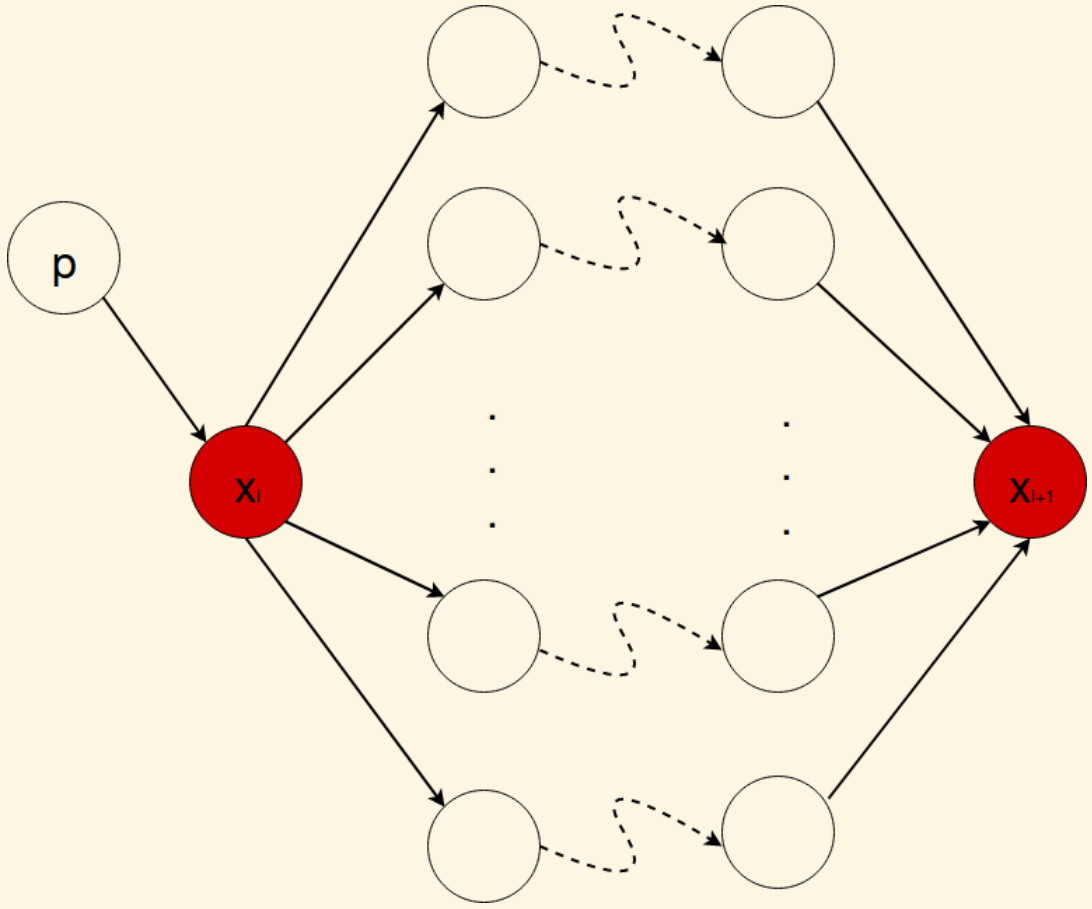


考察

- (現在の頂点, 一つ前の頂点) を状態として dijkstra すれば、
- 頂点数: `O(N^2)`
- 辺数: `O(NM)`
- 一回あたりの dijkstra の計算量は `O(NM log N)`
- 辺の個数回だけ回せばよいので合計 `O(NM^2 log N)`

考察

- (何番目の補給地点まで訪れたか, 直前の頂点) を状態にして DP すれば解けそう



考察

- 状態数: `O(NL)`
- 遷移回数: `O(NM^2L)`
- `L <= 100,000`, `N <= 500`, `M <= 500` -> やばい

考察

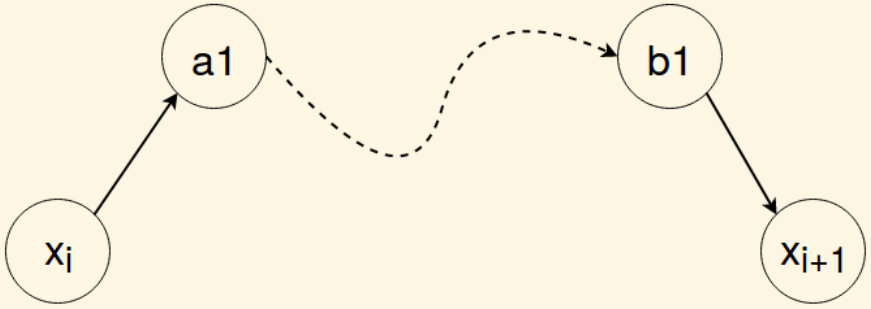
- 遷移数が多すぎてつらい
- だいたい `s -> t` 最短パスを使う
  - 最短パスを使うと直前の頂点に引き返してしまうときや、その後の移動で最短パスを阻害してしまうときには最短パスを使わないかもしれない
- 本当に全ての `(u, v)` について調べる必要がある？

考察

- 答えのパスが
  `x_1 ->...-> p -> x_i ->...-> x_i+1 -> q ->...-> x_L`
  というとき、どんな `p`, `q` であってもちゃんと `x_i -> x_i+1` パスが選択できるようにしつつできるだけ絞りたい

考察

- 最短路 `s -> a1 ->...-> b1 -> t` だけのとき

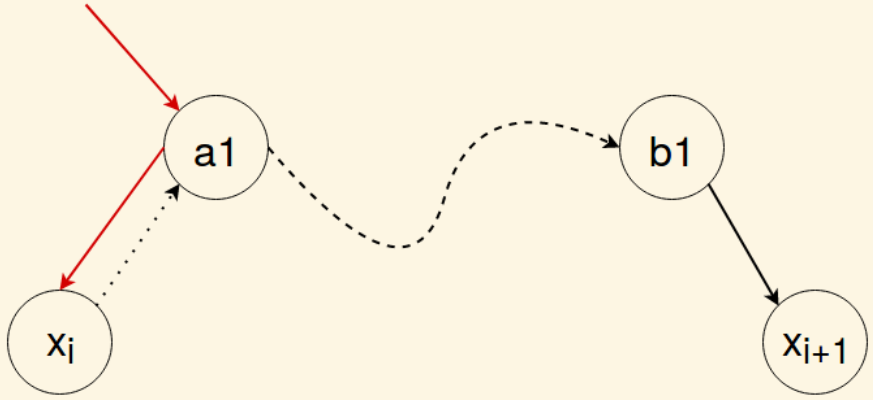


-

-

考察

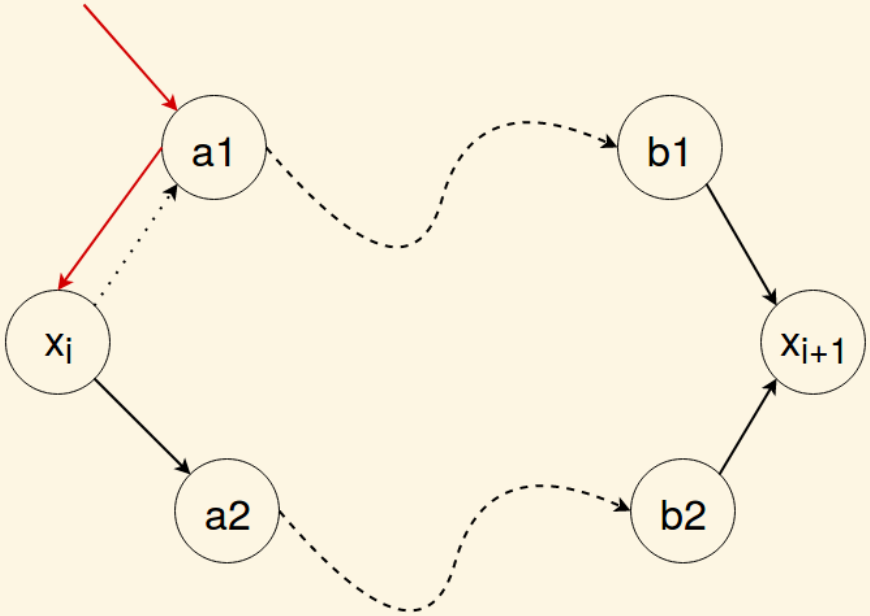
- 最短路 `s -> a1 ->...-> b1 -> t` だけのとき



- `p = a1` のとき死
- `a != a1` となる最短経路も必要

考察

- 最短路 `s -> a2 ->...-> b2 -> t (a2 != a1)` も追加した

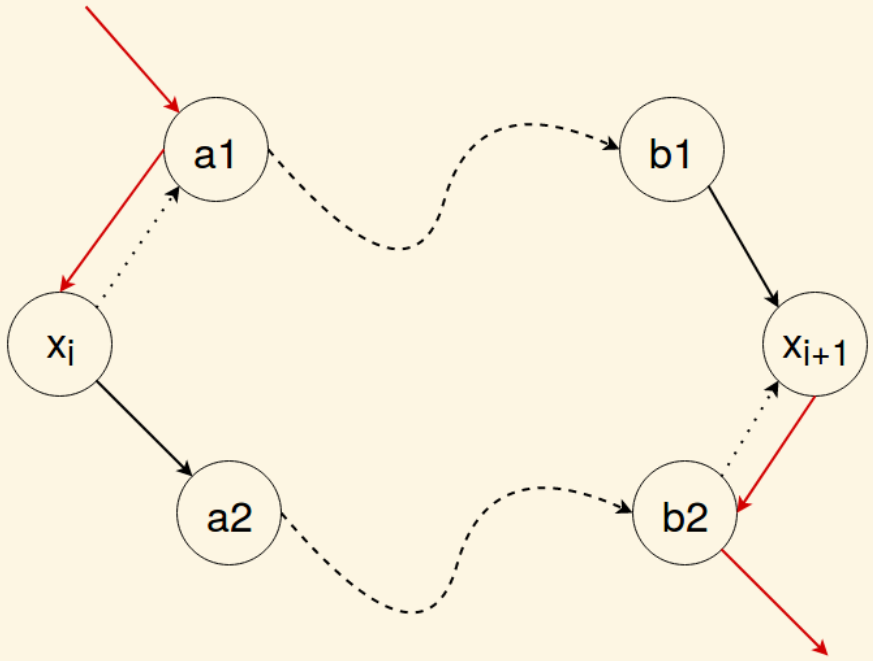


-

-

考察

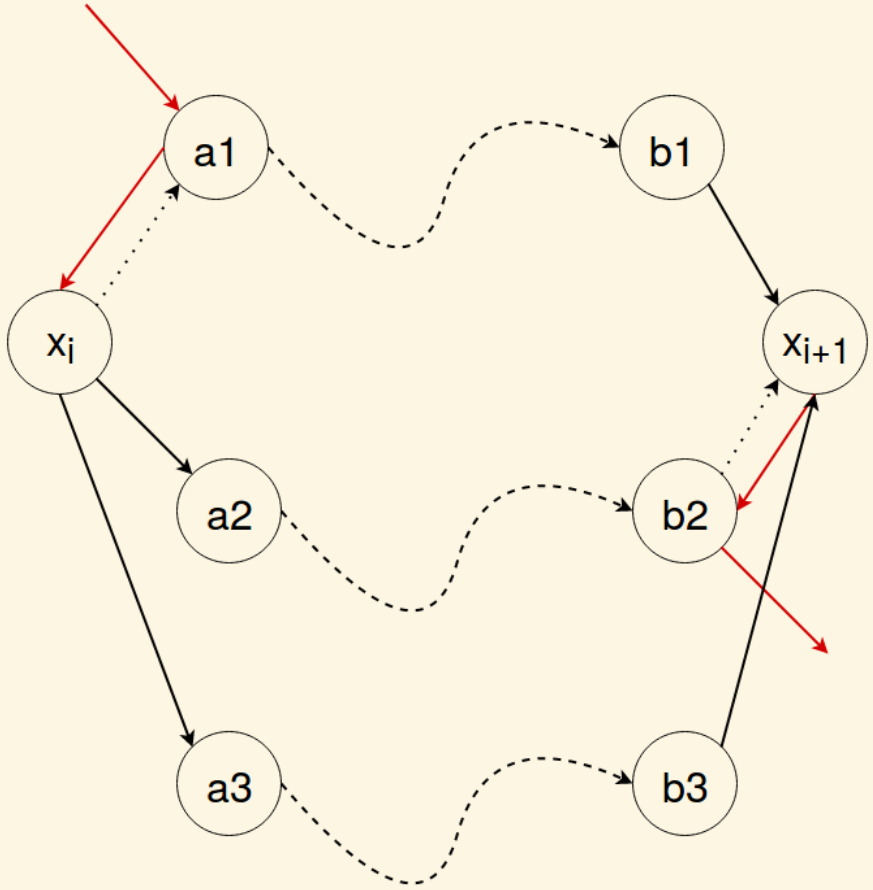
- 最短路 `s -> a2 ->...-> b2 -> t (a2 != a1)` も追加した



- `p = a1, q = b2` のとき死
- `a != a1, b != b2` となる最短経路も必要

考察

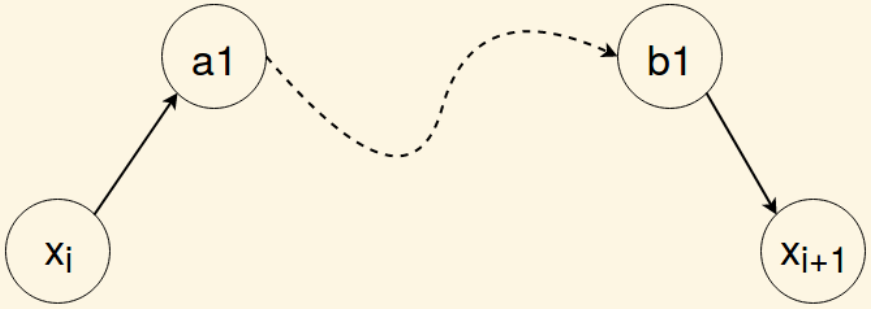
- 最短路 `s -> a3 ->...-> b3 -> t (a3 != a1, b3 != b2)`



- `p = a1` のときは OK になった
- `b != b1` となる経路も必要

考察

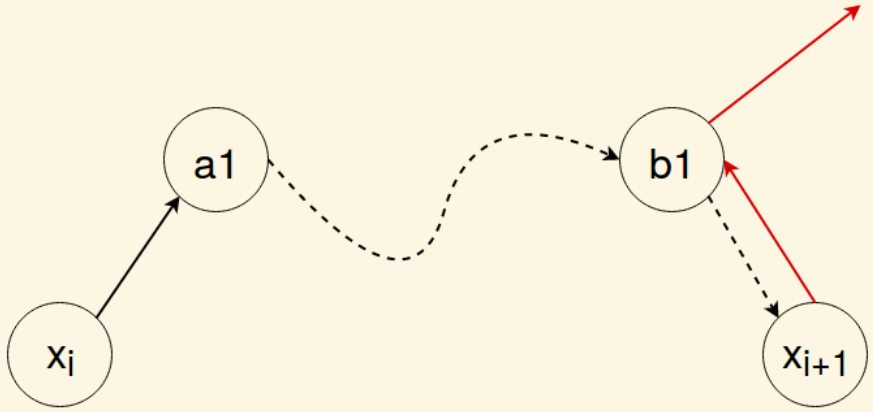
- 最短路 `s -> a1 ->...-> b1 -> t` だけのとき



-
-

考察

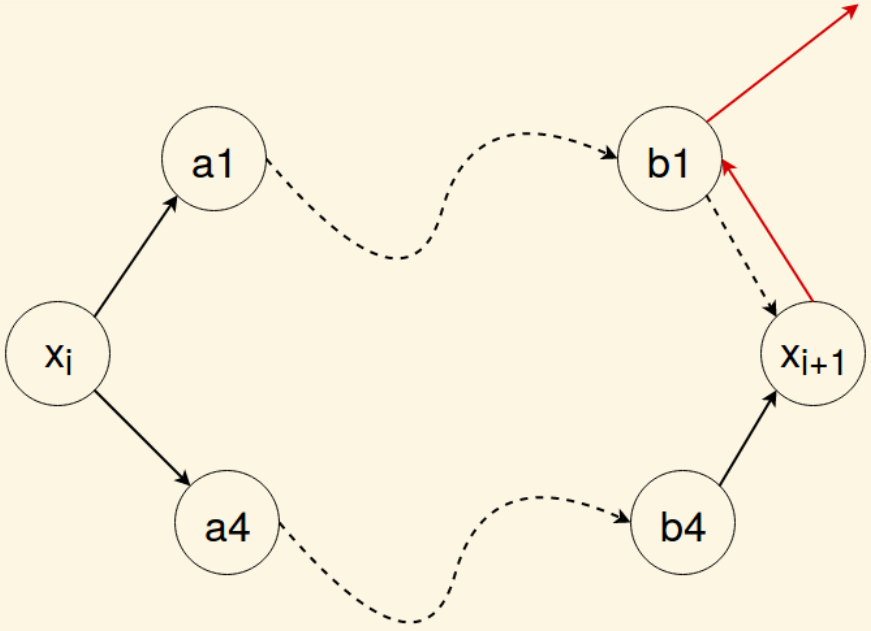
- 最短路 `s -> a1 ->...-> b1 -> t` だけのとき



- `q = b1` のとき死
- `b != b1` となる経路も必要

考察

- 最短路 `s -> a4 ->...-> b4 -> t (b4 != b1)`

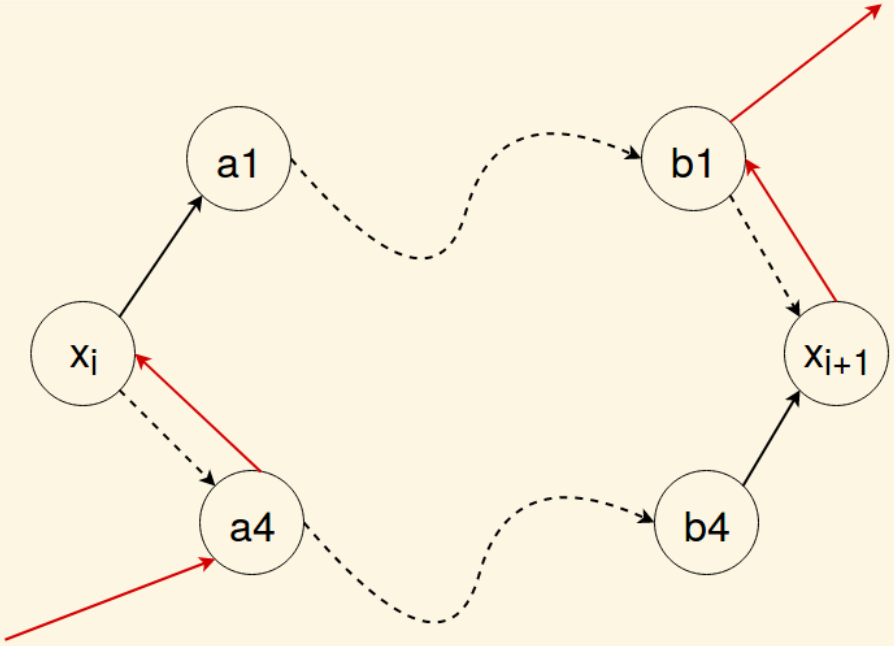


-

-

考察

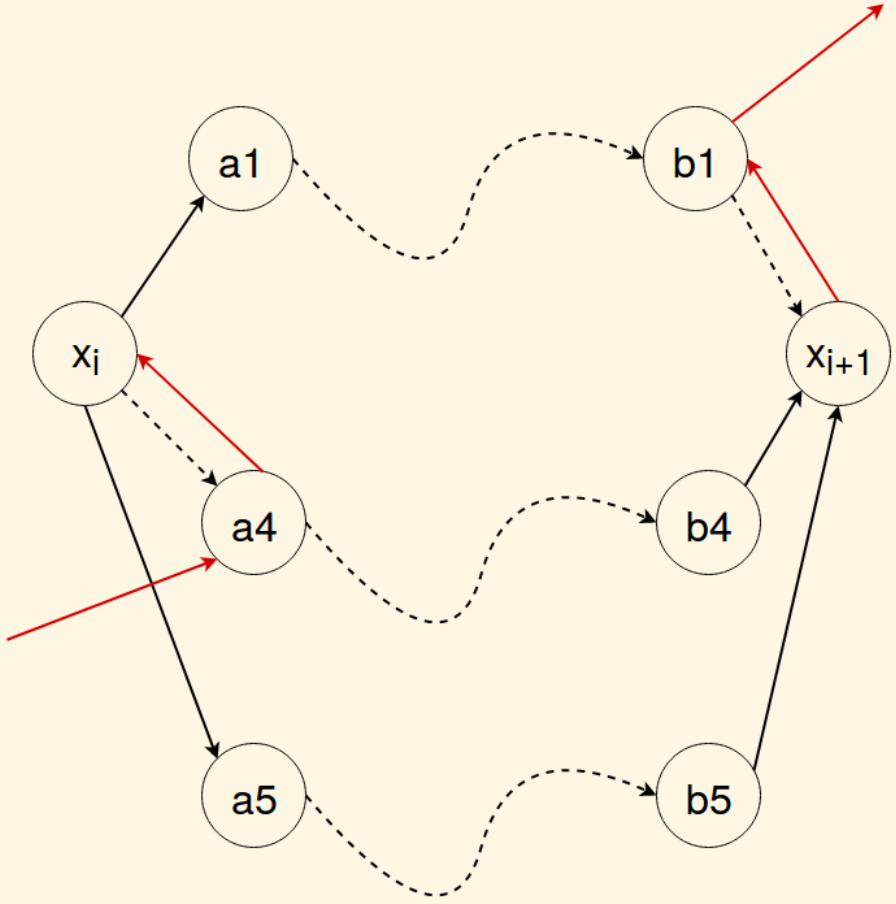
- 最短路 `s -> a4 ->...-> b4 -> t (b4 != b1)`



- `p = a4`, `q = b1` のとき死
- `a != a4, b != b1` となる経路も必要

考察

- 最短路 `s -> a5 ->...-> b5 -> t (a5 != a4, b5 != b1)`



- これで OK
- 合計 `5` 通り

考察

- どんな答えにおいても、`x_i -> x_i+1` パスとして考えられるのは 5 通りしかない

考察

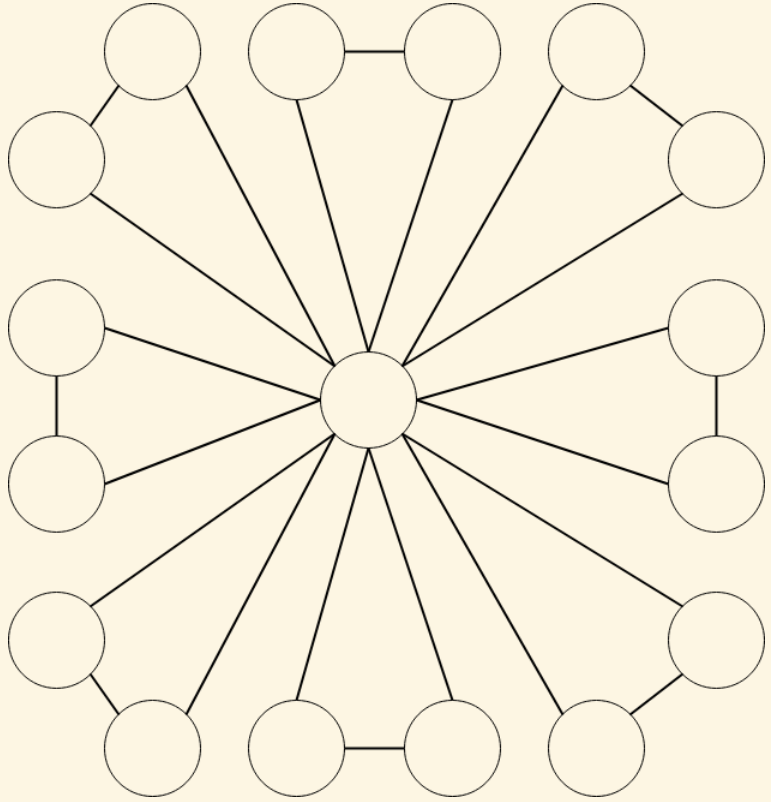
- グッと睨むと
  - `s -> a1 ->...-> b1 -> t`
  - `s -> a2 ->...-> b2 -> t (a2 != a1, b2 != b1)`
  - `s -> a3 ->...-> b3 -> t (a3 != a1, b3 != b2)`
  - `s -> a4 ->...-> b4 -> t (a4 != a2, b4 != b1)`

  の `4` 通りだけでもよいことがわかる
- いずれにせよ、必要なパスを定数個に絞る (以降は `5` 通りに絞ったとする。定数は適宜読み替えてほしい。)

解法

- 補給地点間のパスは 5 (定数) 通りだけ考えればよい
- 前述の (何番目の補給地点まで訪れたか, 直前の頂点) を状態にして DP すれば
- 状態数: `O(NL)`
- 遷移回数: `O(NL)`
- 通る

別解(?)



- こういうグラフがテストケースに無いらしく、普通に毎回 dijkstra でも通るらしい

小課題 3 (15 点, 合計 62 点)

- `T = 1`

考察

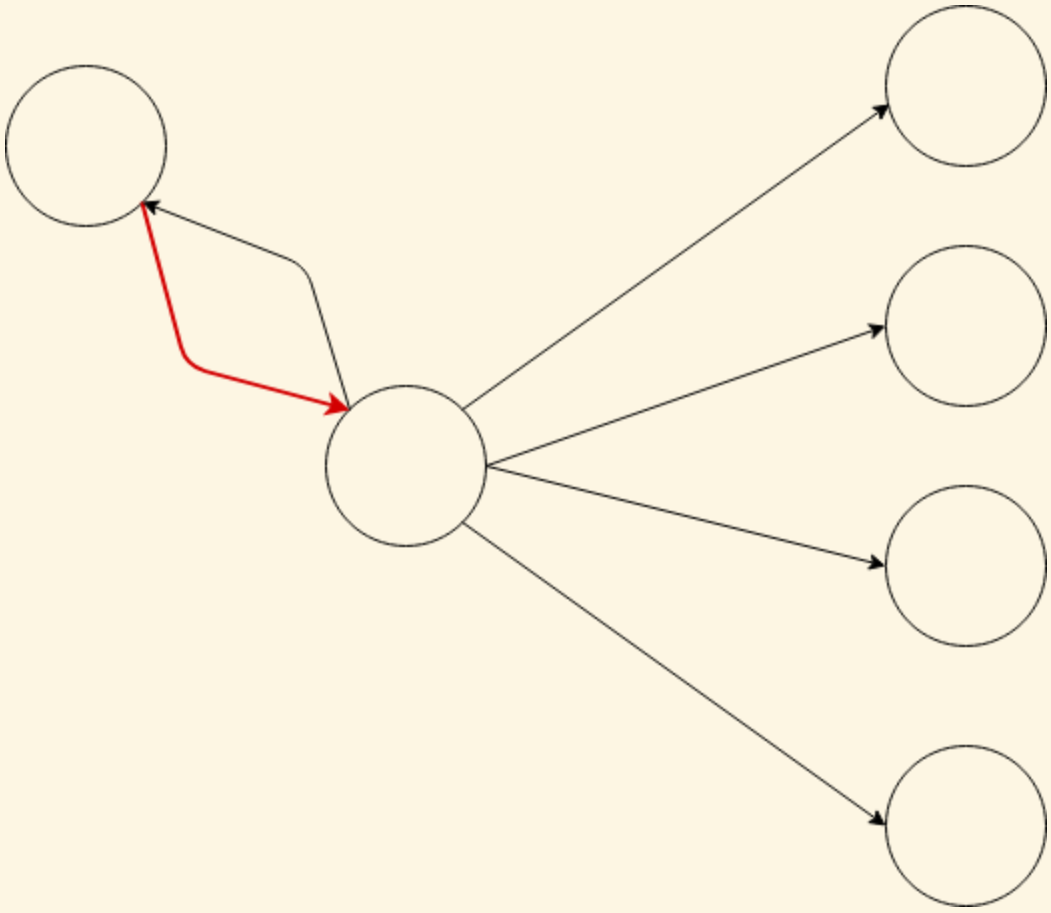
- 頂点数、辺数も `N <= 2000, M <= 2000` とそこそこ多い
- DP 部分の計算量は `O(NL)` なので OK
- `O(NM^2 log N)` かかっていた dijkstra の前計算を高速にする必要がある

考察

- (この辺をこの向きに通るまでの最小時間) を dijkstra で解きたい

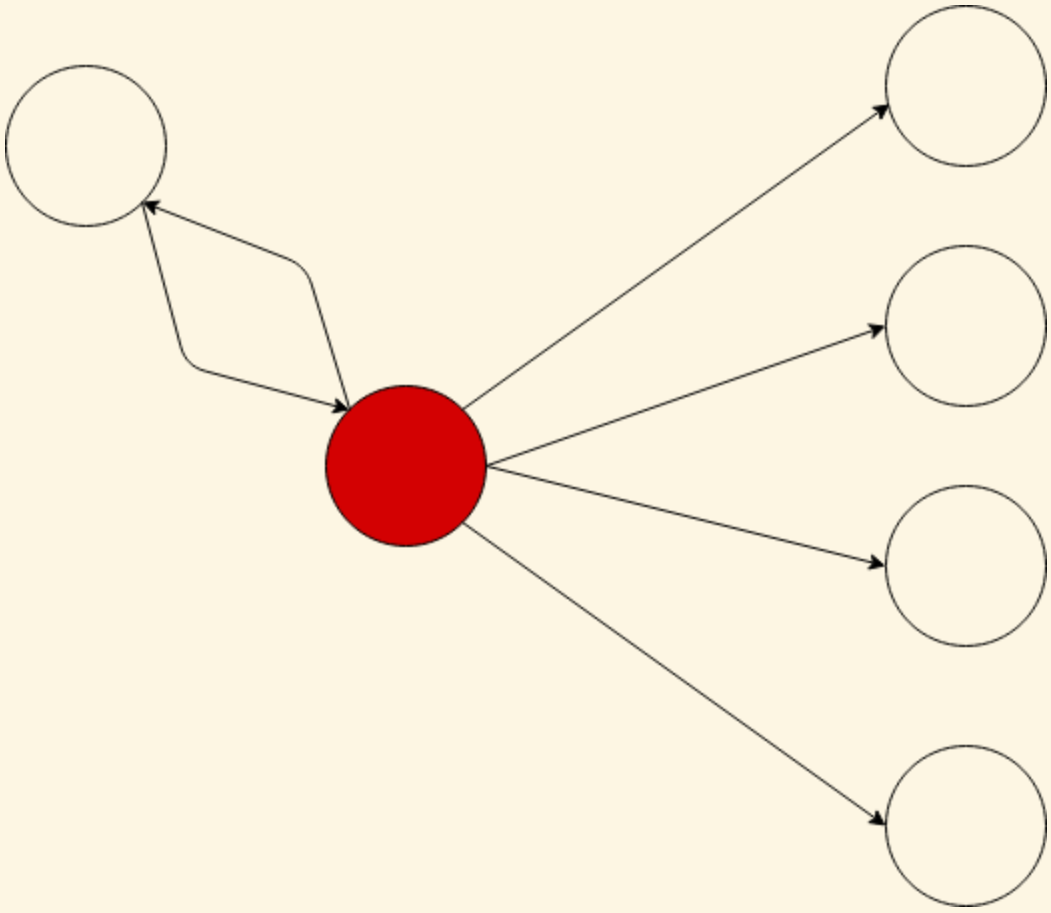
考察

- 頂点を初めて訪れた時、直前の辺の逆辺以外は全部更新する



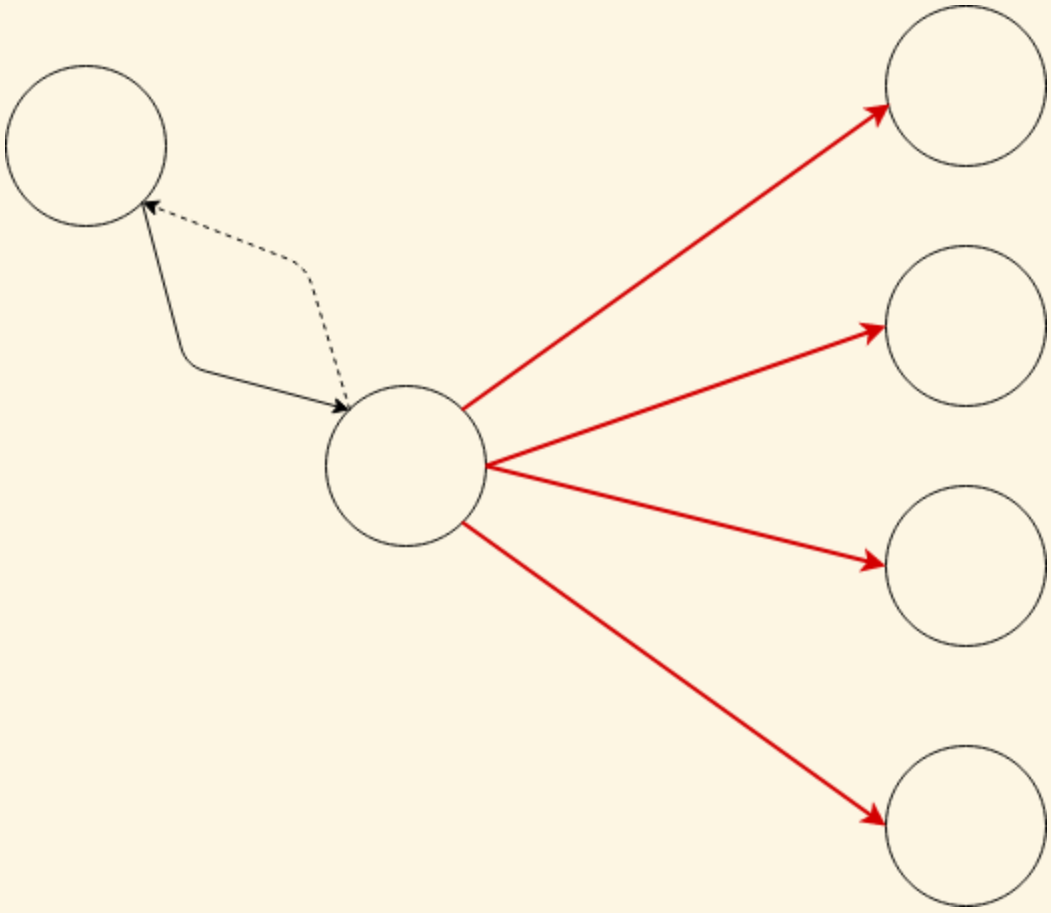
考察

- 頂点を初めて訪れた時、直前の辺の逆辺以外は全部更新する



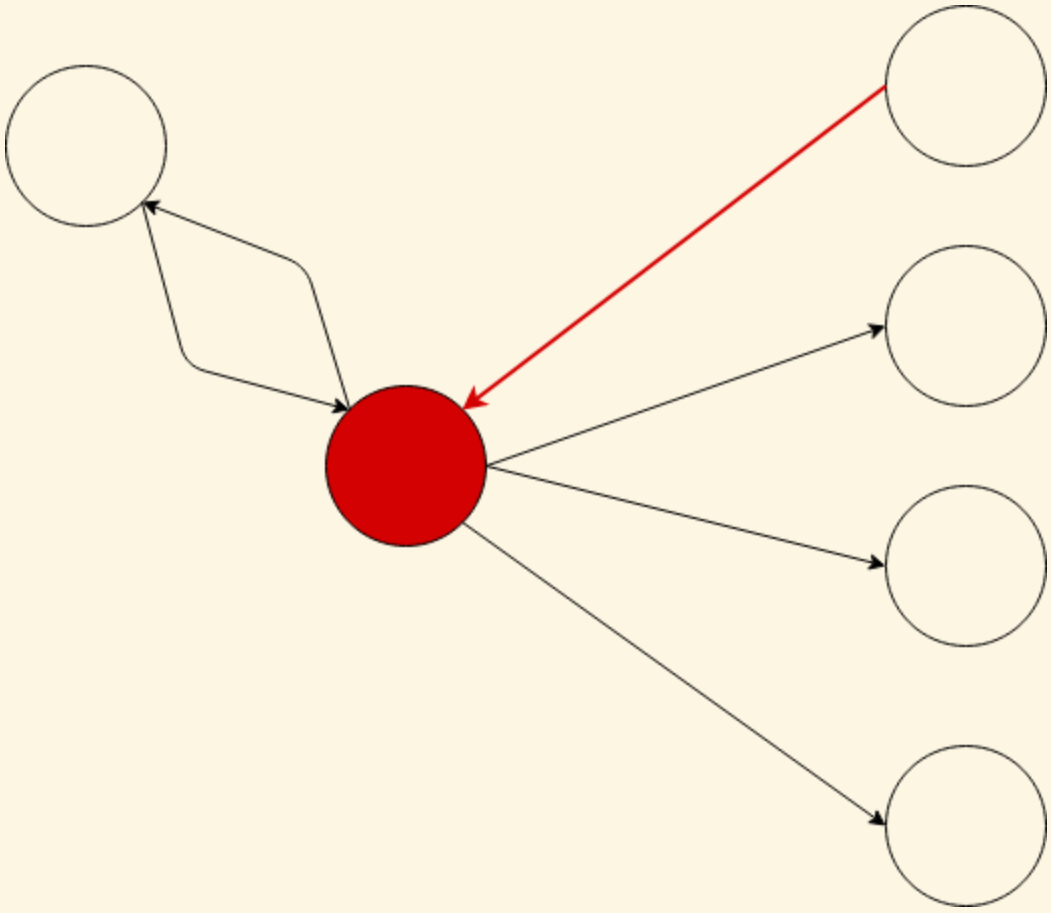
考察

- 頂点を初めて訪れた時、直前の辺の逆辺以外は全部更新する



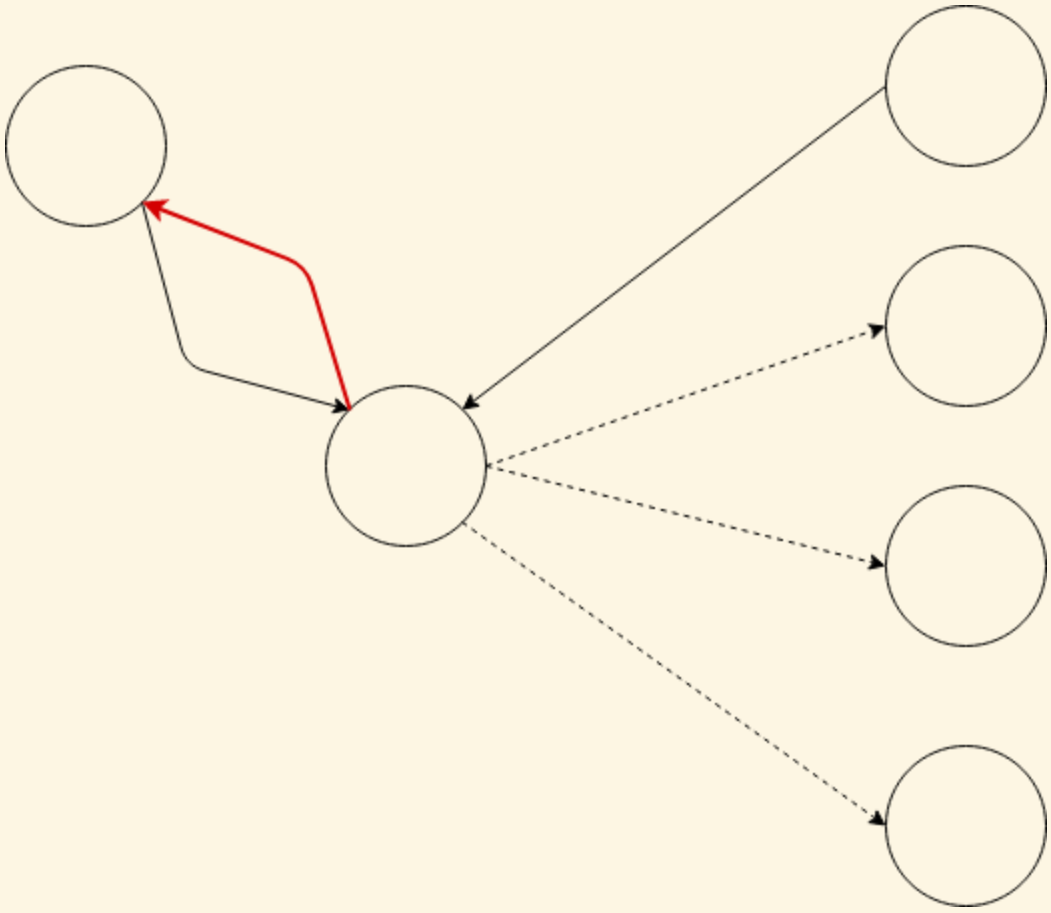
考察

- 次別の方向から来ると、初めて来た方向のみ更新すれば十分



考察

- 次別の方向から来ると、初めて来た方向のみ更新すれば十分



解法

- 辺を状態として dijkstra
- 頂点に (まだ訪れていない) or (最初どの辺から来たか) をもっておく
- 初めて来たら、直前に使った辺の逆辺以外は全部更新する
- 次別の方向から来ると、初めて来た方向のみ更新する
- 一回の dijkstra が `O(M log M)`
- これを `2M` 回やれば良いので `O(M^2 log M)`
- 間に合う

小課題 4 (38 点, 合計 100 点)

- 追加の制約は無い

クエリが `T <= 100,000` 回来る

考察

- dijkstra の `O(M^2 log M)`
- 毎回 DP に `O(NL)` かけると合計 `O(TNL)` で間に合わないので高速に処理する必要がある

考察

- 変更される補給地点の数は 1 日ごとに高々 `1` 箇所
- ほとんど同じ計算を `T` 回することになっている
- 補給計画の変更しない区間は使いまわせないか

こういう一点更新クエリたくさんみたいなものを処理するテクといえば...?

Segment tree

解法

- 区間の状態として必要なのは最初使う辺と最後使う辺（間で何が使われているかは範囲外には影響しない）
- 先ほど考察したように、補給地点間のパスとして考えられるのは高々 `5` 通り
- Segtree の `[l, r)` に対応するノードに、`5 x 5` 行列 `node[i, j]` := `l` 番目の補給地点から `r-1` 番目の補給地点に行くのの最小値、ただし最初のパスとして、`x[l] -> x[l+1]` 間の `i` 番目のパスを使い `x[r-2] -> x[r-1]` 間の `j` 番目のパスを使うを格納する

解法

- `[l, r)`, `[r, s)` をマージするとき、`5^4` 通りのパスを考えて同様の `5 x 5` 行列を構築すればよい

解法

- 前計算の dijkstra: `O(M^2 log M)`
- Segtree の初期化 `O(L)`
- クエリの回答: `O(T log L)`

考察

- 定数倍が若干重い `5^4`
- 実は `5 x 5` 通りのパスを保存する必要はない
- さきほど補給地点間のパスとして `5` 本しか考えなくてよいと考察したことと同様の考察をすると各ノードに `5` 通りのパスしか保存しなくてもよい
- これで定数倍も安心
- 最初からパスを `4` 通りに絞っているとここで頑張らなくても安心

得点分布

```
100:
62: ##
47: ##
12: #######
0 : #########
```

ご清聴ありがとうございました