いろはちゃんコンテスト Day2 解説

この pdf は、いろはちゃんコンテスト Day2 の各問題の解説をまとめたものです。個別の解説はそれぞれの問題ページ下部にあります。

いろはちゃんコンテスト 2019 day2 わ-わたのはら 解説

settyan117

2019/05/01

真・p 字決まりの条件を考察してみましょう。S が真・p 字決まりであるとは、

- S のどの長さ q の部分列も、歌集の他の歌の部分列でない

を満たす最小の q が p であるということでした。

これを和歌 $S,\ T$ のみが含まれる歌集について言い換えると、

- 長さ $p-1$ の S の部分列には T の部分列でもあるものが存在するが、長さ p では存在しない

となり、p は $S,\ T$ に共通の部分列のうち最も長いものの長さに 1 を足したものである、とわかります。よって、$S,\ T$ に共通する最長の部分列を求めることを考えます。

複数の文字列に共通する最長の部分列のことを LCS (Longest Common Subsequence) と呼び、その中でも 2 つの文字列に対する問題は、2 つの文字列の長さの積に比例する時間で解けることが知られています。具体的には、動的計画法 (DP) を用います。

大きさ $(N+1)\times(N+1)$ の 2 次元配列 lcs について、$lcs[i,\ j]$ を

- S の先頭から i 文字目までと、T の先頭から j 文字目までの LCS の長さ

と定義すると、$i,\ j>0$ において、$lcs[i,\ j]$ への遷移は次の 2 パターンが考えられます。

(i) **S の i 文字目と T の j 文字目が等しい場合** $lcs[i,\ j]=lcs[i-1,\ j-1]+1$

(ii) **S の i 文字目と T の j 文字目が異なる場合** $lcs[i,\ j]=\max\{lcs[i-1,\ j],\ lcs[i,\ j-1]\}$

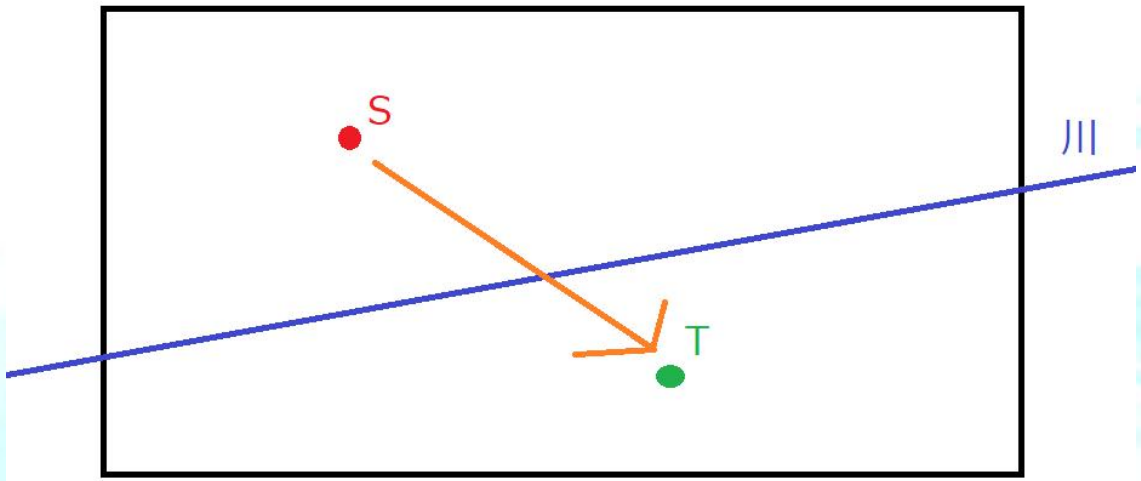
片方が空文字列の場合、LCS は空文字列で長さは 0 なので、$lcs[i,\ 0]$ と $lcs[0,\ j]$ はすべて 0 で初期化します。そののちに、適切な順番に上の遷移を行えば lcs を埋めることができ、S と T の LCS の長さは $lcs[N,N]$ になります。この値に 1 を足すと p の値が求まり、これでこの問題が解けました。

河川敷の変態仮面 解説

physics0523

問題概要

- 長方形を直線の川が貫いている
- へんなひとがSからTまで一直線に走ってくる
- へんなひとが川を飛び越えるかどうか判定



幾何です

お察しの通り、この問題は幾何です

「実装めんどくさそう」などと思わずがんばりましょう

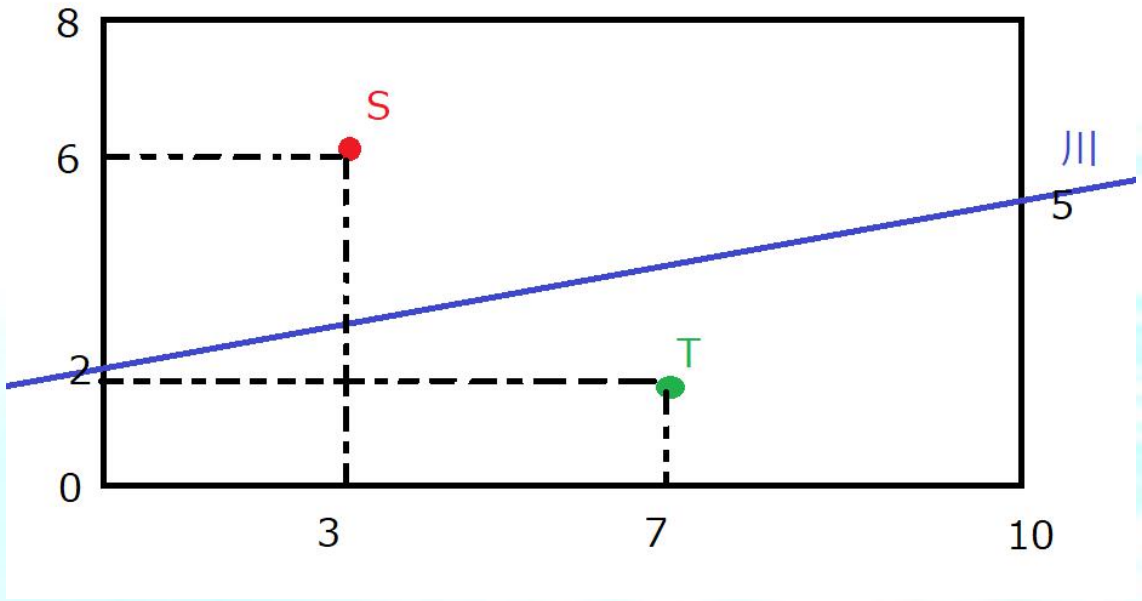
(実はこの問題はかなりシンプルに実装出来ます)

解法はいろいろありますが、最も汎用性のあるものを紹介します

「二次元ベクトルの外積」

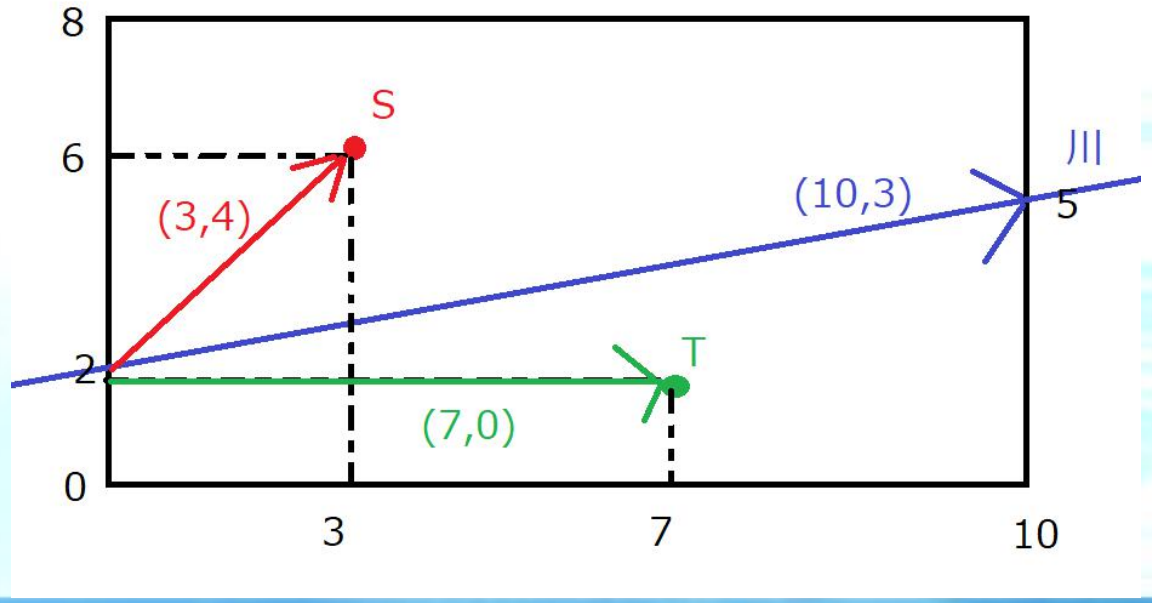
具体的に以下の設定として説明する

(ここからの説明はしばしば「二次元ベクトルの外積」または「符号付き面積」などと説明されます、詳しくは検索してください)



3本の「矢印」を考える

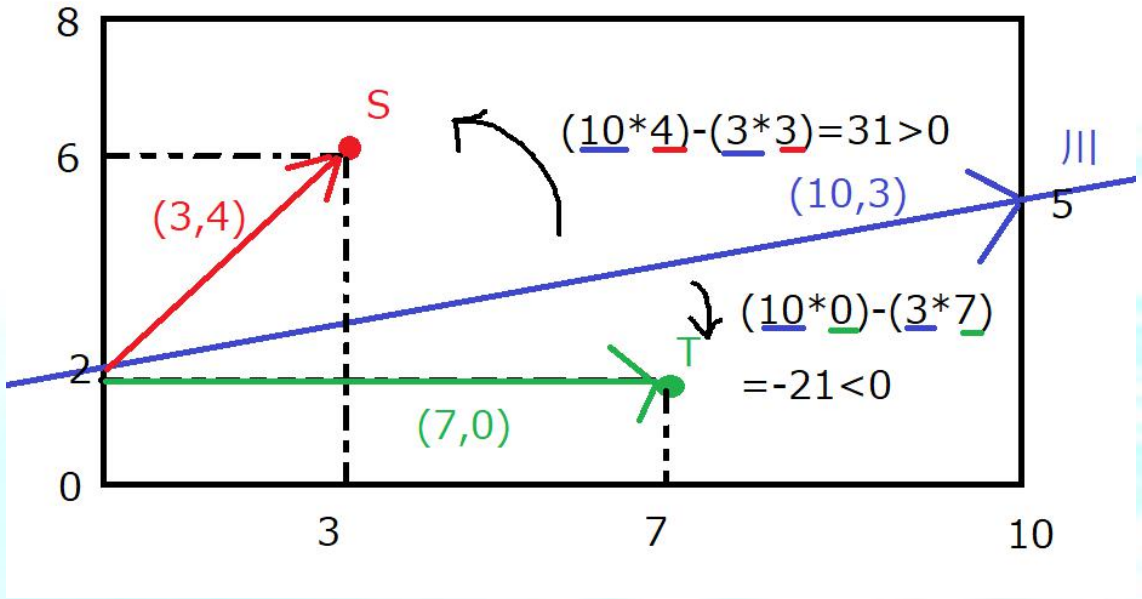
図のように3本の矢印を考えて、X方向にa移動するごとにY方向にb移動することを(a,b)と表現する(a,bは負になってもよい)



基準の矢印から時計回りか反時計回りか

基準の矢印(a,b)からある矢印(c,d)が時計回り方向か反時計回り方向のどちらにあるかは、(a*d)-(b*c)で計算できる

これが正なら時計回り方向、負なら反時計回り方向



実際に交差するかの判定

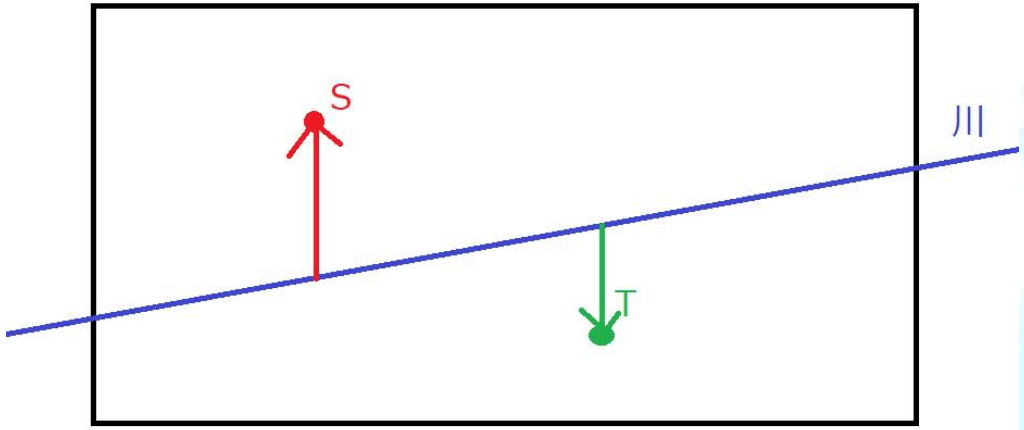
もし、青を基準に赤の回転方向と緑の回転方向が違えば、SからTまで一直線に移動するときに川を跨ぐことになる

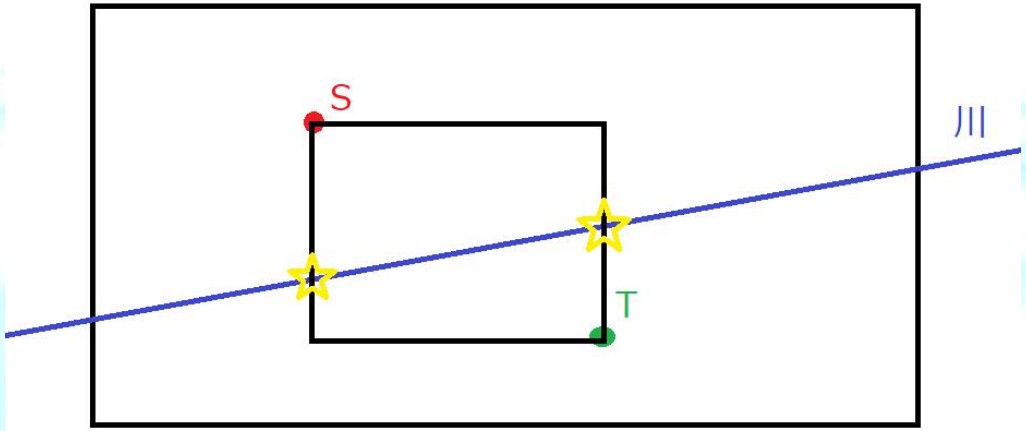
逆に、回転方向が同じならSからTまで一直線に移動するときに川を跨がずに済む

他の解法

(左)S,Tのあるx座標で川が通過するy座標を計算して川の上側か下側かを判定(誤差に注意)

(右)S,Tを対角とする長方形を考えて川がどの辺を横切るか判定(実装も場合分けも重くなります)





正解率、FA

AC/Trying:

268/300(89.3%)

FA:beet(04:21)

この操作は座標圧縮と呼ばれる手法です。

連想配列を使う方法が最も簡単です。たとえば、C++では map というものが用意されています。次の手順で解いてみましょう。

1. map に、$A_1, \cdots, A_N$ をキーとして追加する。
2. map ではキーが小さい順にソートして保管されているので、小さいほうから順に $1, 2, \cdots$ と値を割り当てる。
3. 圧縮後の値を求めるには、map で A_i に割り当てられた値を参照する。

いろはちゃんコンテスト解説

Day2-D:楽しすぎる家庭菜園

@Pro_ktmr

注意

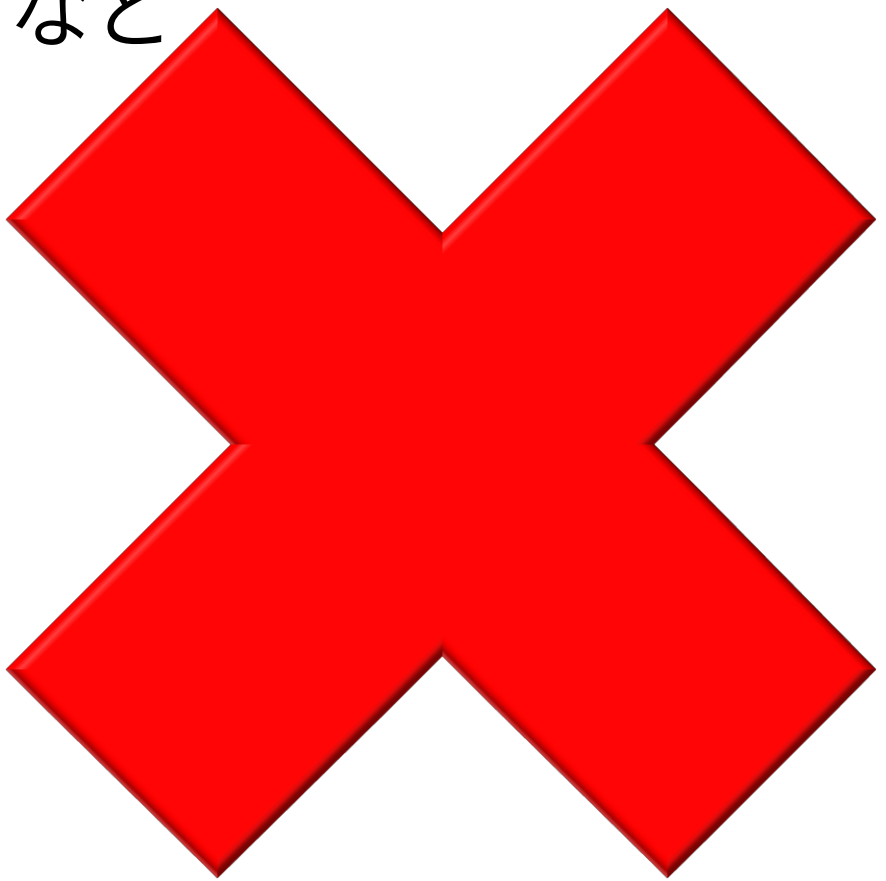
- 内容の正確性には十分注意していますが、間違った情報が含まれることもあります。何かお気づきのことがあればご連絡ください。
- このスライドのサンプルコードはC++で実装されています。

この問題の狙い

- フローが使えるか　など

この問題の狙い

- フローが使えるか　など



問題文が途中までかなりフローっぽい

この問題の狙い

- 最小全域木を求められるか　など

問題概要

- N頂点M辺の連結な無向グラフがある
- 辺にはバラバラのコストが定まっている

- 連結な状態を保って辺の本数を最小化
- 辺のコストの総和を最大化

- $N \leqq 200000$　$M \leqq 400000$

考察

- 連結な状態を保って辺の本数を最小化すると・・・木になる！
- いわゆる全域木を求める問題

考察

- 連結な状態を保って辺の本数を最小化すると・・・木になる！
- いわゆる全域木を求める問題

- 辺のコストを最大化・・・

考察

- 連結な状態を保って辺の本数を最小化すると・・・木になる！
- いわゆる全域木を求める問題

- 辺のコストを最大化・・・最小化の逆とも考えられる
- よって、最小全域木を求める問題！(のちょっと変なアレンジ)

余談

- どうして最小全域木という言葉はあるのに最大全域木という言葉は一般に使われないか
- 普通コストの和を最大化したければわざわざ木にする必要がないからだと思います、たぶん

最小全域木の求め方

- この解説ではクラスカル法を紹介する

- UnionFindを用意する(連結判定用)
- 辺をコストの降順でソートする
- 辺をコスト大きい方から見ていって、頂点A_iとB_iが非連結ならその辺を採用する
  連結ならその辺は採用しない

- こんなに簡単な貪欲法でOK(証明は検索してください)

解法① クラスカル法(逆順ver) O(M log M)

```
#include"bits/stdc++.h"
using namespace std;

struct UnionFind{
private:
	vector<int> parent;
	vector<int> rank;
public:
	void init(int N){
		parent.clear();
		rank.clear();
		for(int i=0; i<N; i++) parent.push_back(i);
		for(int i=0; i<N; i++) rank.push_back(0);
	}
	int root(int a){
		if(parent[a] == a) return a;
		return parent[a] = root(parent[a]);
	}
	void unite(int a, int b){
		int rootA = root(a);
		int rootB = root(b);
		if(rootA == rootB) return;
		if(rank[rootA] < rank[rootB]) parent[rootA] = rootB;
		else{
			parent[rootB] = rootA;
			if(rank[rootA] == rank[rootB]) rank[rootA]++;
		}
	}
	bool same(int a, int b){
		return root(a) == root(b);
	}
};

struct Edge{
	int a, b, i;
	long long c;
	bool operator<(const Edge& e) const {
		return (c < e.c);
	}
};

int N, M;
vector<Edge> v;
UnionFind uf;
vector<int> ans;
int main(){
	scanf("%d%d", &N, &M);
	v.resize(M);
	for(int i=0; i<M; i++){
		scanf("%d%d%d", &v[i].a, &v[i].b, &v[i].c);
		v[i].i = i+1;
	}
	sort(v.begin(), v.end());
	uf.init(N+1);
	for(int i=M-1; i>=0; i--){
		if(!uf.same(v[i].a, v[i].b)){
			uf.unite(v[i].a, v[i].b);
			ans.push_back(v[i].i);
		}
	}
	sort(ans.begin(), ans.end());
	for(int i=0; i<N-1; i++) printf("%d¥n", ans[i]);
}
```

Thank you for reading!

連呼 解説

Writer : ミドリムシ

問題概要

次の条件を満たす文字列の個数を**mod 10^9+7**で求めなさい

- **A**が**N**個と**B**が**M**個からなる
- 最初が**A**で最後が**B**
- **AAA**を部分文字列に含む

考察

「**AAA**を部分文字列に含む」は考えづらいので、
「**AAA**を部分文字列に含まない」文字列の個数を数えてみる

最後に **A**から始まり**B**で終わる文字列の個数($_{N+M-2}C_{M-1}$) から引けば良い

AAAを部分文字列に含まない文字列にはどういう性質があるか？

考察

AAAを部分文字列に含まない文字列は
　AABB…BBABB…BBAABB…BB
のような形になる(**A**が**1-2**個と**B**が**1**個以上を繰り返した形)

これを**A**と**B**の境目で区切ると次のような文字列の列になる
　AA, BB…BB, A, BB…BB, AA, BB…BB

考察

この文字列の列は次のような性質を持っている
・左から奇数番目は**A**だけが**1-2**個ある文字列
・左から偶数番目は**B**だけの文字列
・列の長さは偶数で、**2**以上**min(N, M)**以下
・左から奇数番目の文字列長の和は**N**
・左から偶数番目の文字列長の和は**M**

逆に、これらの条件を満たしていれば、そこから**AAA**を部分文字列に含まない文字列が構築できる

考察

文字列の列の長さを全探索することを考える

　文字列の列の長さを**2k**と固定すると、次の問題に変換できる

Q. 長さ**2k**の正整数の列で、左から奇数番目の値の総和が**N**、左から偶数番目の値の総和が**M**であるもののうち、左から奇数番目の値が**1**か**2**のものはいくつか？

左から奇数番目と偶数番目は独立してるから分けて考えられそう
→答えは 奇数番目の場合の数×偶数番目の場合の数 になる！

(積の法則**)**

左から奇数番目パート

左から奇数番目だけ抜き出すと次のようになる

Q. 長さ**k**の数列で、各要素が**1**か**2**であり、かつ総和が**N**であるものはいくつか？

数列の要素を**1**小さくして考えると、${}_{k}C_{N-k}$個であることがわかる

(ただし、${}_{a}C_{b}$は**b<0**または**a<b**の場合には**0**とする 以降も同様**)**

左から偶数番目パート

左から偶数番目だけ抜き出すと次のようになる

Q. 長さ**k**の正整数列で総和が**M**であるものはいくつか？

数列の要素を**1**小さくして考えると、重複組み合わせになる
よって、${}_{k}H_{M-k}={}_{M-1}C_{M-k}$個であることがわかる

まとめ

AAAを部分文字列に含まない文字列の個数は、 **k=1, 2, …**
floor(min(N, M)/2) と動かした時の ${}_{k}C_{N-k} \times {}_{M-1}C_{M-k}$ の総和になる

AAAを部分文字列に含む文字列の個数は、${}_{N+M-2}C_{M-1}$ からこれを引いた値

[Q.E.D.]

実装

${}_nC_k$の計算は いろはちゃんとマス目 **(ABC042-D)** のようにすると
前計算**O(**添字の最大値**)**時間、クエリ**O(1)**時間で出来る

(いろはちゃんとマス目 解説：arc058.contest.atcoder.jp/data/arc/058/editorial.pdf)

今回は添字の最大値が**O(N+M)**なので、計算量も**O(N+M)**になる

総入れ替え 解説

physics0523

問題概要

IROHATHM(一体なんのゲームでしょうか…)をするために次のようなゲームをする(ゲームをするためのゲーム…)

・箱Aに100円玉A_1枚、50円玉A_2枚を詰める

・箱Bに100円玉B_1枚、50円玉B_2枚を詰める

・箱Cに100円玉C_1枚、50円玉C_2枚を詰める

物理好きさんとひらきちさんが交互に「好きな箱を選んでコインを1枚ランダムに引く」を繰り返す

双方が最終的な所持金を最大化するように動くとき、物理好きさんの所持金の期待値を求めよ

ところで

100円玉と50円玉…そもそも間違えますか?
中央のあたりに穴があるかどうかで見分けられると思います

そのへんはお口チャックで進めましょう

制約を見てみよう

$$A_1, A_2, B_1, B_2, C_1, C_2 \leq 10$$

なにやらおぞましい制約

少し無茶な解法もできそうということを念頭に入れて考えていく

状況を少し整理

このゲームは物理好きさんとひらきちさんの2人で行っており、二人が獲得する金額の総和は変わらない

とすると、次のことが成立する

「ひらきちさんの所持金を最大化する戦略」は「物理好きさんの所持金を最小化する戦略」と同じ

ゲームには再帰DPが便利!

ゲームを考えるとき、再帰DPを考えるとうまくいくことがあり、結論から言うと、今回もそれでうまく行きます

ではどのような再帰DPを考えればいいのでしょうか…?

方針

func(状況)を与えてそれ以降に物理好きさんの獲得する金額の期待値が返ってくる関数が欲しい

(状況とは、全ての箱についてのそれぞれコインの残り枚数と、今どちらの手番か)

これは作れます!!

(ちなみに、全ての箱についてのそれぞれコインの残り枚数が決まればどちらの手番かは確定します)

再帰DPの構造

疑似コードっぽく記述すると、

func(状況){

if(同じ状況を既に計算している){その結果を返す}

[[手番を持つ人が最善の戦略となる箱からコインを引く]]

この状況の結果を保存して返す

}

[[]]の部分が遷移となり、本質です(次ページ以降で詳説)

遷移の仕方(1)

もし手番をもつ人が箱A(100円玉p枚、50円玉q枚)からコインを引いたとする

すると、

確率p/(p+q)で箱Aのみから1枚だけ100円玉が減った状況

確率q/(p+q)で箱Aのみから1枚だけ50円玉が減った状況

に遷移する

遷移の仕方(2)

すると、遷移先の状況についての期待値はfunc(状況)を再帰的に呼ぶことで計算できる

最終的に、再帰は全ての箱が空になる状況まで行きつく

箱B,Cも同様

ここで、物理好きさんが手番なら、期待値最大となる箱を選べばよい

ひらきちさんの手番なら?→「物理好きさんの期待値を最小化」なので期待値最小の箱を選べばよい

結果の保存「メモ化」

同じ状況について再帰関数を何度も呼んでいちいち計算しているとかなり重くなりますが、そのような場合、計算結果を保存しておけます(これをメモ化と言います)

今回の場合、

double dp[Aに残った100円][Aに残った50円][…(箱B,Cも同様)

={それ以降に物理好きさんの獲得する金額の期待値}

のように保存でき、制約よりこの6次元配列はMLEしません!

(考えようによっては少し無茶な解法)

正解率、FA

AC/Trying:

81/88(92.0%)

FA:beet(26:25)

いろはちゃんコンテスト解説

Day2-G:通学路

@Pro_ktmr

注意

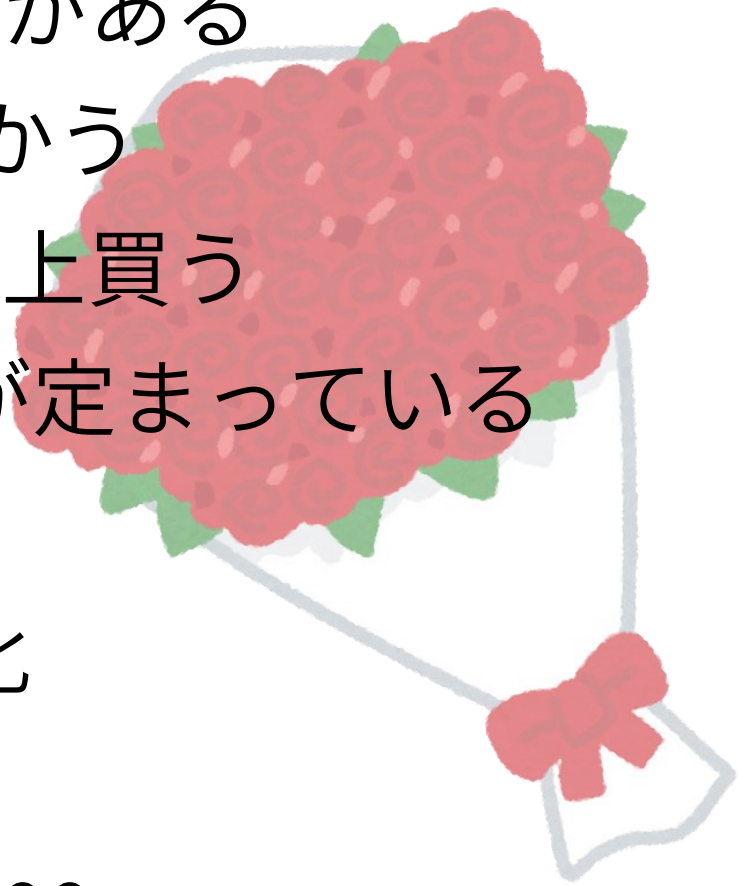
- 内容の正確性には十分注意していますが、間違った情報が含まれることもあります。何かお気づきのことがあればご連絡ください。
- このスライドのサンプルコードはC++で実装されています。

この問題の狙い

- ダイクストラ法が実装できるか
- 頂点を拡張することができるか　など

問題概要

- 駅と鉄道路線(グラフ)がある
- 頂点1から頂点Nへ向かう
- 道中で花を合計K本以上買う
- 路線、花にはコストが定まっている



- コストの合計を最小化

- N,K ≦ 1000　M ≦ 2000

考察

- キーワード「グラフ」「最小」といえばダイクストラ法
- しかし、ダイクストラ法では、何本花を買っているかの情報を持つことができない

考察

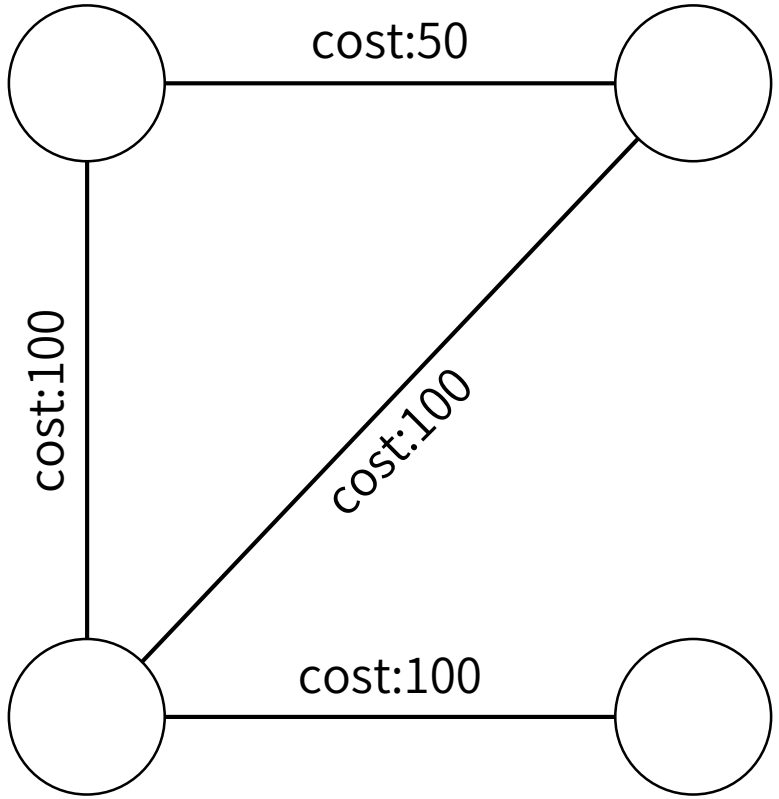
- キーワード「グラフ」「最小」といえばダイクストラ法
- しかし、ダイクストラ法では、何本花を買っているかの情報を持つことができない？

考察

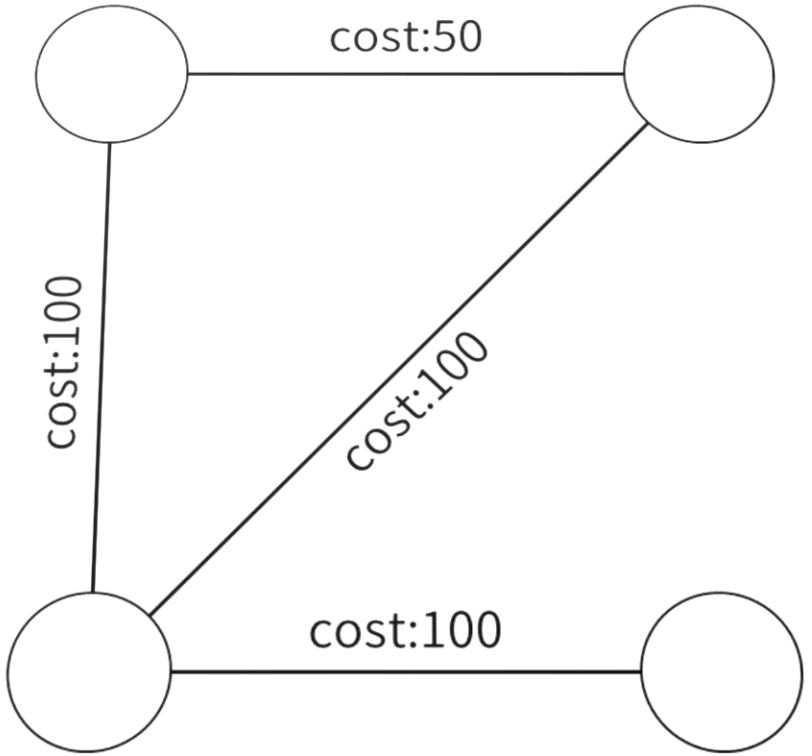
- キーワード「グラフ」「最小」といえばダイクストラ法
- しかし、ダイクストラ法では、何本花を買っているかの情報を持つことができる

- 頂点を増やしてしまえばよい

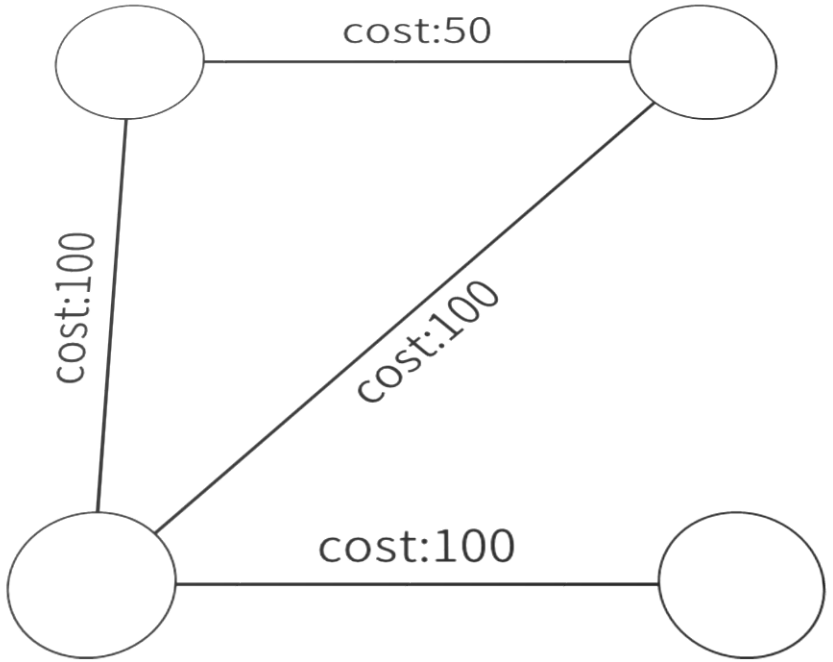
頂点の拡張：入出力例1



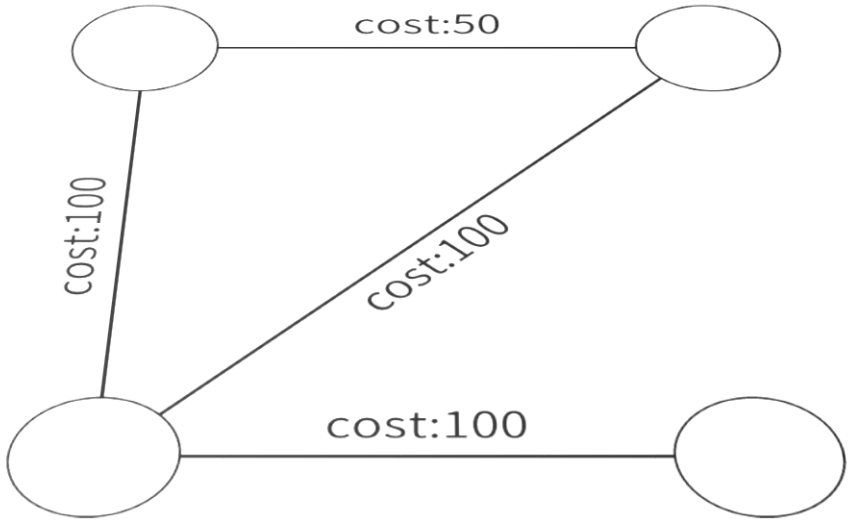
頂点の拡張：入出力例1



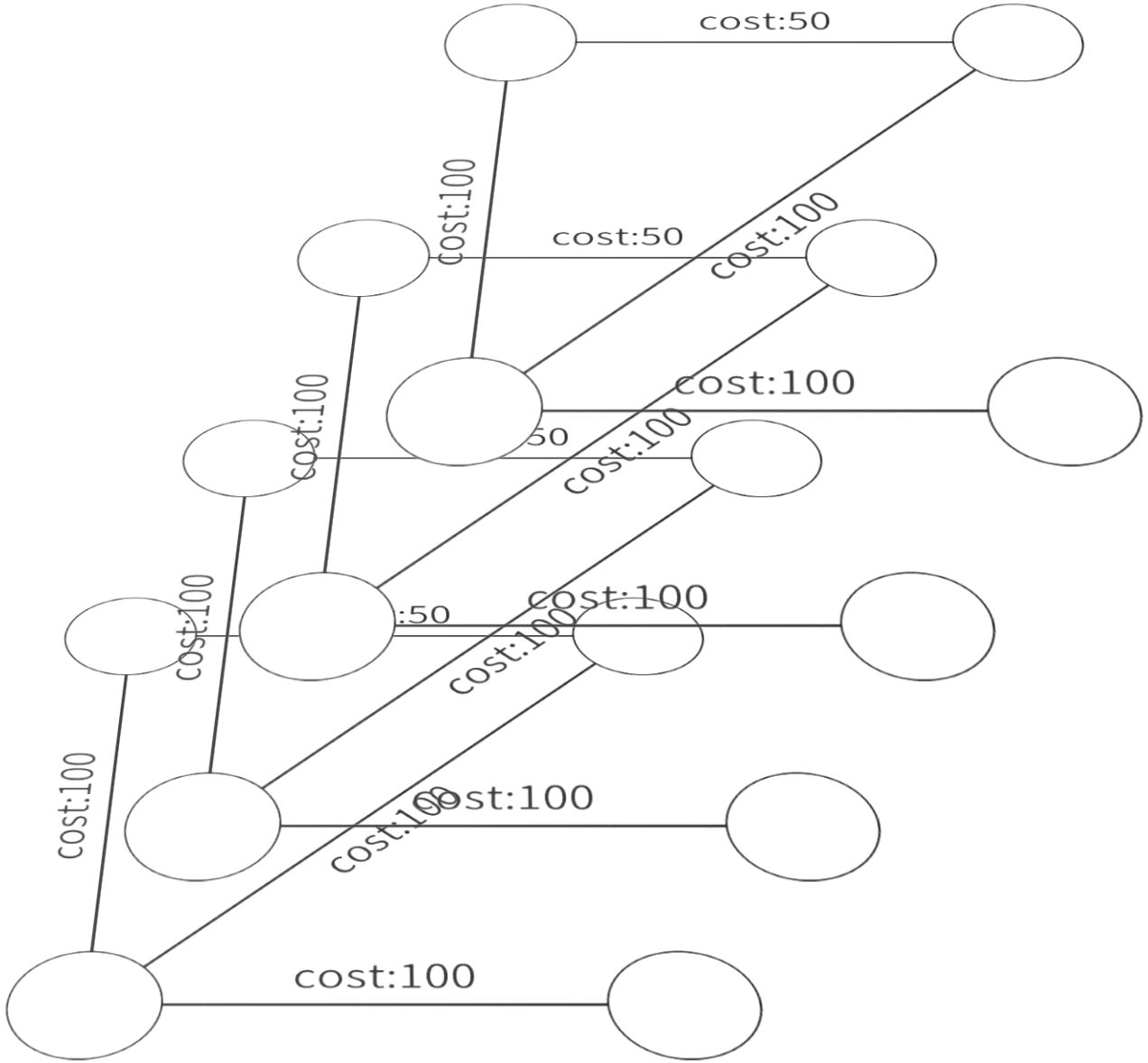
頂点の拡張：入出力例1



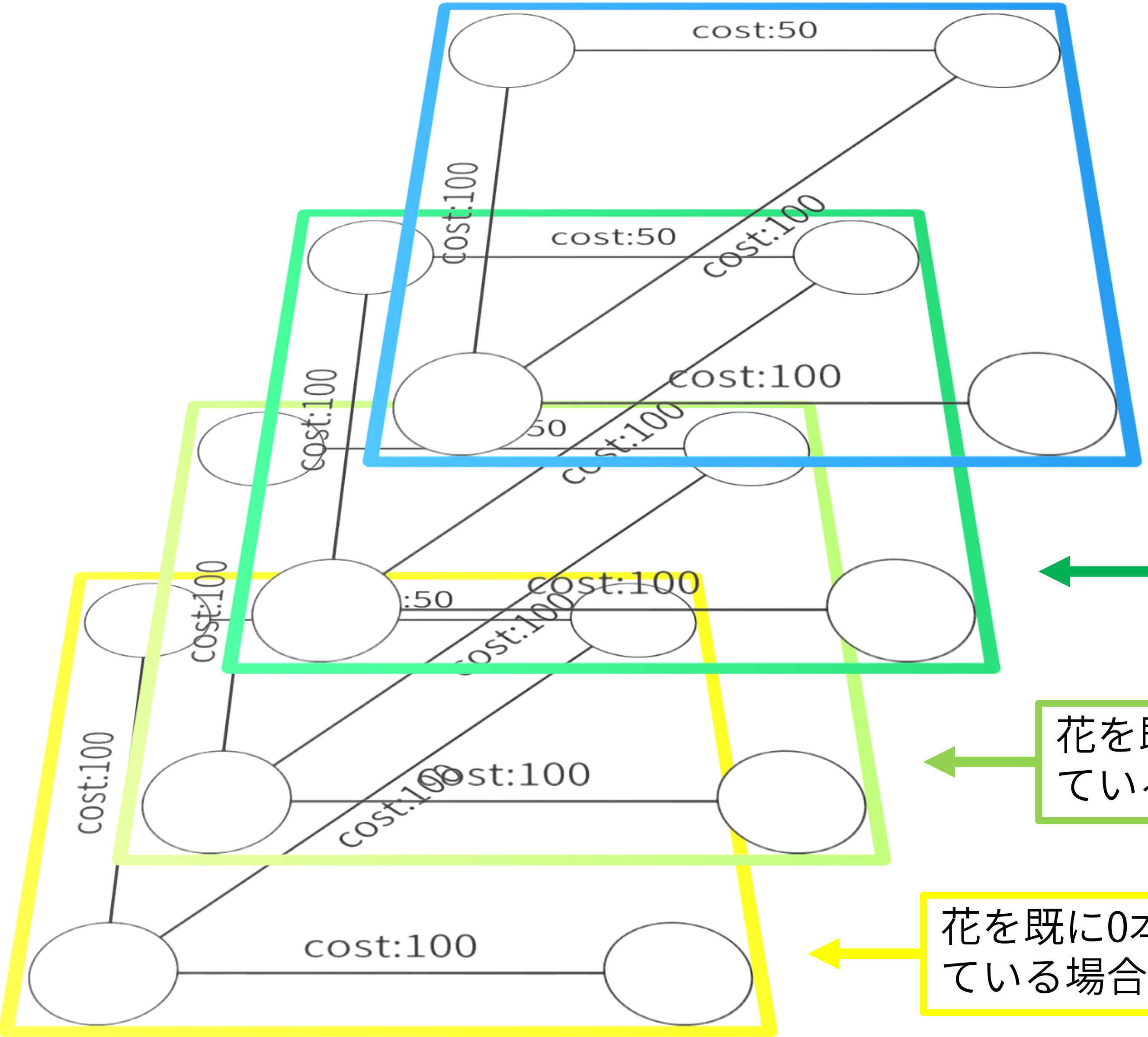
頂点の拡張：入出力例1



頂点の拡張：入出力例1


cost:50
cost:100
cost:50
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100
cost:100

頂点の拡張：入出力例1



花を既に3本買っている場合の層

花を既に2本買っている場合の層

花を既に1本買っている場合の層

花を既に0本買っている場合の層

考察

- このように頂点を拡張すると
- 頂点数はK倍または2K倍になる
  →2NK個 $\leqq 2\times10^6$
- 辺数もK倍になり、上下を接続する辺が最大で2NK本増える
  →MK+2NK本 $\leqq 4\times10^6$

- ダイクストラ法の時間計算量はO(ElogV)だから、この拡張を行っても十分高速

余談

- 頂点を拡張してダイクストラ法を行うことを、拡張ダイクストラ法と呼ぶこともある
- ダイクストラ法を拡張しているような誤解を招くことから、この表現を嫌う人も多い
- 結局このテクを何と呼ぶかはTwitterでも未だに議論が盛んである

解法① 頂点を拡張したダイクストラ法

```
#include"bits/stdc++.h"
using namespace std;
const long long INF = 1LL<<62;
int N, M, K, X[2000];
long long Y[2000];
vector<pair<int,long long>> edge[2000];
long long cost[2000][4000];
int main(){
        cin >> N >> M >> K;
        for(int i=0; i<M; i++){
                int A, B;
                long long C;
                cin >> A >> B >> C;
                A--; B--;
                edge[A].push_back(make_pair(B, C));
                edge[B].push_back(make_pair(A, C));
        }
        for(int i=0; i<N; i++){
                cin >> X[i] >> Y[i];
                for(int j=0; j<2*K; j++) cost[i][j] = INF;
        }
        cout << dykstra() << endl;
}
```

```
long long dykstra(){
        priority_queue<pair<long long, pair<int, int>>> que;
        cost[0][0] = 0; que.push({ 0,{0,0} });
        while(!que.empty()){
                long long c = -que.top().first;
                long long sta = que.top().second.first;
                long long flo = que.top().second.second;
                que.pop();
                if(flo < K && cost[sta][flo+X[sta]] > c + Y[sta]){
                        cost[sta][flo+X[sta]] = c + Y[sta];
                        que.push({ -
cost[sta][flo+X[sta]] ,{sta,flo+X[sta]} });
                }
                for(int i=0; i<edge[sta].size(); i++){
                        if(cost[edge[sta][i].first][flo] > c +
edge[sta][i].second){
                                cost[edge[sta][i].first][flo] = c +
edge[sta][i].second;
                                que.push({ -
cost[edge[sta][i].first][flo] ,{edge[sta][i].first,flo} });
                        }
                }
        }
        long long answer = INF;
        for(int i=0; i<K; i++) answer = min(answer, cost[N-1][K+i]);
        if(answer == INF) answer = -1;
        return answer;
}
```

Thank you for reading!

H – 根室の巫女

sheyasutaka

問題概要

- 文字列の各接頭辞のlongest borderが与えられる
- それに合致する文字列があるか判定し，あれば1つ構築せよ

解説

- Moris-Pratt法をします
- j>Bi のとき s[i] != s[j]と記録
- j=Bi のとき s[i] = s[j] とする(この時上に矛盾しないか見る)
  - その後break
- j<Bi のとき No

- j=Bi=0 になったときは新しい文字としてよいことが示せる

な - 南極 (800 点)

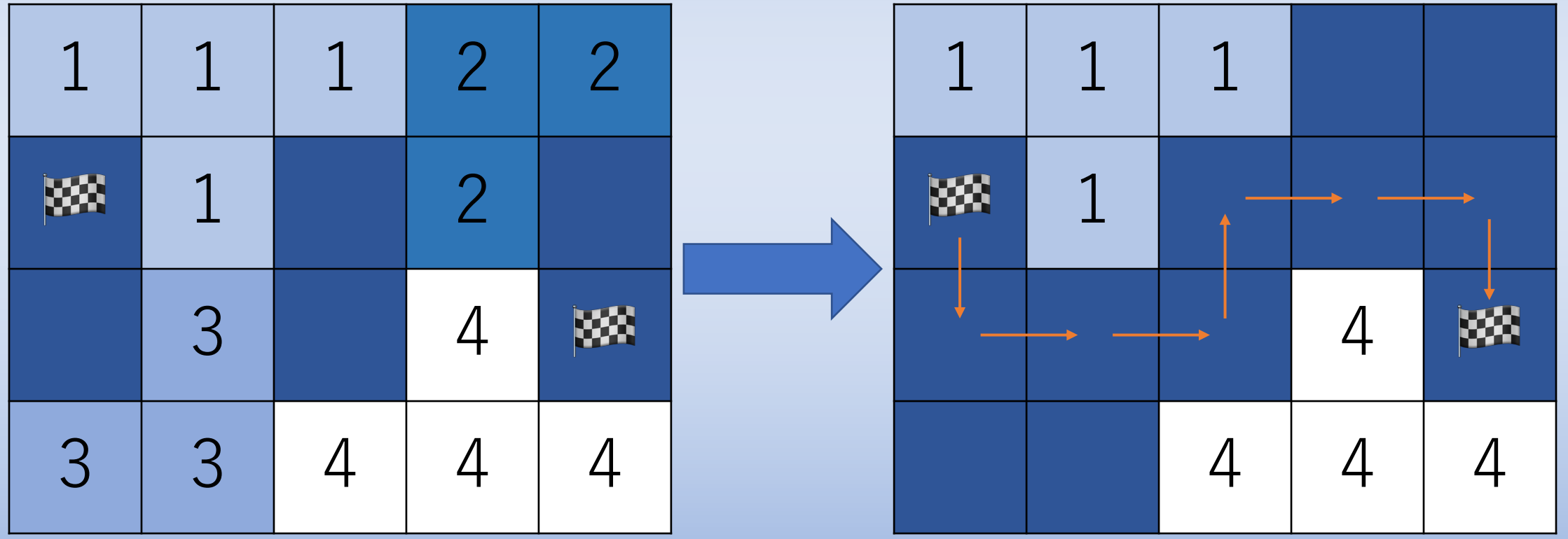
IROHACON 2019 DAY 2 解説

問題概要

- $H \times W$ のマス目であらわされる南極海
- X 個の連結な氷山がある
  - それぞれの氷山を解かすコストは $C_1, C_2, \ldots, C_X$
- マス (i, j) は氷山 $A_{i,j}$ に属している
  - $A_{i,j} = 0$ のとき何にも属していない
- スタートからゴールまでの最小コストは？

入力例 1

- 入力例 1 は次のようになります (コスト 5+8=13)



部分点 1 (160 点)：$H, W \leq 40, X \leq 9$

- 氷山の個数 X が 9 個以下
- どの氷山を消すか・消さないかは 2^X 通りあるから、これを全探索できる
- このときに (sx, sy) から (gx, gy) まで連結になっているか判定するのは、深さ優先探索で $O(HW)$ で解ける
- よって、全体の計算量が $O(2^X \times HW)$ なので実行時間制限に間に合う！

部分点 2 (560 点)：$H, W \leq 40$

- 南極海の大きさ $H, W \leq 40$
- ここからは別の方針を考えなければならない
- 例：氷山の大きさが全部 1×1 のとき、この問題は最短経路問題に帰着できて、ダイクストラ法で解ける！

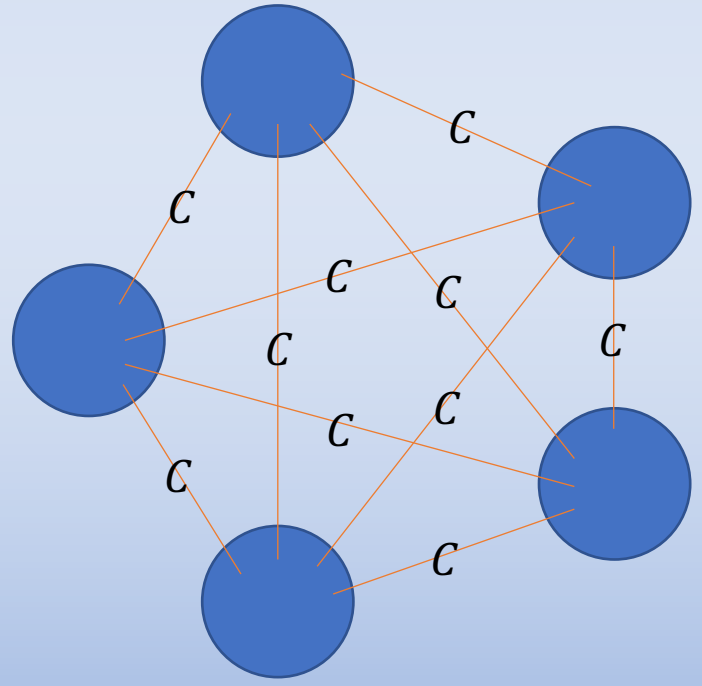
→ この問題も最短経路問題に帰着させてダイクストラ法で解きたい・・・

部分点 2 (560 点)：$H, W \leq 40$

- ある氷山について、「境界線」から「境界線」まで行くのにかかるコストが C_i と考えればよい
- そうすると最短経路問題に帰着できそう
- 境界線は最大で $O(HW)$ 個あるので、$O(HW)$ 頂点 $O(H^2W^2)$ 辺の最短経路問題を解くことになる
- $O(V^2)$ のダイクストラ法を使って、全体の計算量 $O(H^2W^2)$ で解ける

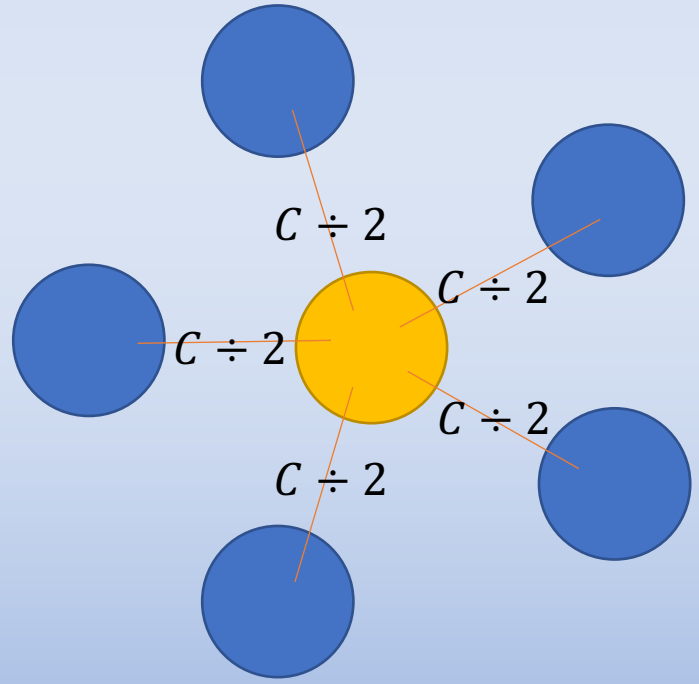
満点解法 (800 点)：$H, W \leq 500$

- 1 個の氷山に対して、もともとは次のようなグラフの辺の張り方をしていた



満点解法 (800 点)：$H, W \leq 500$

- これをこうすればよい！



満点解法 (800 点)：$H, W \leq 500$

- そうすると、$O(HW)$ 頂点 $O(HW)$ 辺の最短経路問題を解くことになって、ダイクストラ法で $O(HW \log HW)$ で解ける

- 実装は・・・？かなり重め

満点解法 (800 点)：別解

- 氷山以外のマスも、連結成分は 1 つの「コスト 0 の氷山」としてみなす
- これの「氷山を拡張したもの」をグラフの 1 頂点として、地図上で隣接する氷山をグラフの辺とする
- グラフ上で、頂点を通るときにコスト C_i がかかり、頂点 s から頂点 g まで移動するのにかかる最短コストは？

満点解法 (800 点)：別解

- これはダイクストラ法で解くことができる
- 連結成分の数を R としたとき
  - グラフを作るのに $O(HW)$
  - ダイクストラ法で解くのに $O(R \log R)$
- よって、全体の計算量は $O(HW + R \log R)$ で解けた
- こっちは実装が比較的簡単

J

sheyasutaka

解説

- セグ木です

- 以下の4つの値を持つ
  - 計算結果
  - 左端から×の連続する塊の値
  - 右端から×の連続する塊の値
  - ＋を持つかのbool
- がんばればmergeできる

いろはちゃんコンテスト解説

Day2-K:虫取り

@Pro_ktmr

注意

- 内容の正確性には十分注意していますが、間違った情報が含まれることもあります。何かお気づきのことがあればご連絡ください。
- このスライドにサンプルコードはありません。
  https://atcoder.jp/contests/iroha2019-day2/submissions/5202558
  こちらをご覧ください。

独り言

- だんだんきたむーと架空の彼女がイチャイチャする問題設定が適当になっていってる気がする・・・
- 個人的には虫が嫌いなので、なぜデートで虫取りをしているのか意味が分かりません

- まさか最終問題を担当させてもらえるとは思ってなかったので、とても嬉しいです
- もしよければTwitter等で感想を聞かせてください

独り言

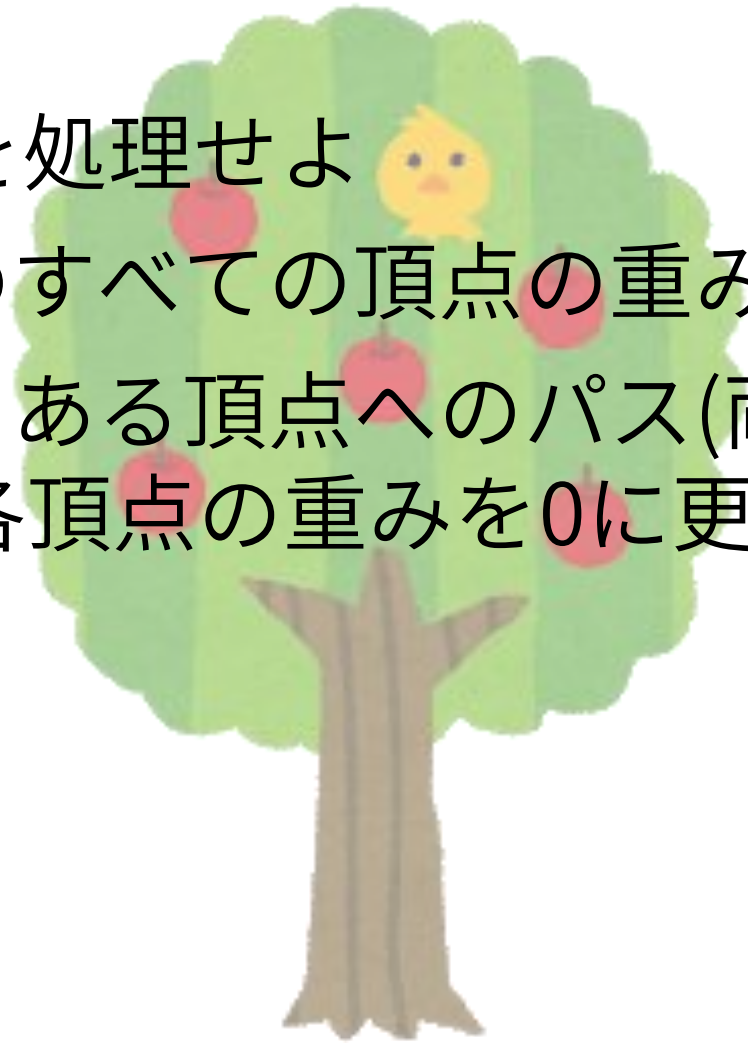
- なんでサンプルコードを載せてないの？
- 長いからです。全体で250行あります。
- JOIに出すはずの問題を間違えてAtCoderで出してしまったかもしれません(そんなことはないです)

この問題の狙い

- 遅延セグ木が書けるか
- 木構造の特徴を知っているか
- LCAは求められるか
- 部分木クエリに対応できるか(DFS木)
- パスクエリに対応できるか(HL分解)
- 両方同時にできるか　など

問題概要

- N頂点の木がある
- Q回のクエリ(2種類)を処理せよ
- クエリ1:ある部分木のすべての頂点の重みを加算
- クエリ2:ある頂点からある頂点へのパス(両端含む)上の重みの和を出力し、各頂点の重みを0に更新
- $N \leqq 200000$
- $Q \leqq 80000$



木について

- 木の基本についてはきたむーさんという超賢い方がわかりやすくスライドにまとめてくださっているので、そちらを先にご覧ください
- https://www.slideshare.net/Proktmr/ss-138534092
- ここでは最短パスの一意性、DFS木の作り方、EularTourとLCAの関係などについて書かれています(これらがわかる人は読まなくていいです)

考察

- 問題を整理する

- クエリA：部分木に対する加算
- クエリB：パスの和とパスに対する更新

- 各クエリが$O(\log N)$ぐらいでできると嬉しい

ところで

- この問題の木が直線、つまり数列の問題なら？

ところで

- この問題の木が直線、つまり数列の問題なら？
- 部分木もパスも直線

- 部分木に対する加算：RAQ
- パスの和：RSQ
- パスに対する更新：RUQ

ところで

- この問題の木が直線、つまり数列の問題なら？
- 部分木もパスも直線

- 部分木に対する加算：RAQ
- パスの和：RSQ
- パスに対する更新：RUQ

- これらが全部処理できるデータ構造・・・

ところで

- この問題の木が直線、つまり数列の問題なら？
- 部分木もパスも直線

- 部分木に対する加算：RAQ
- パスの和：RSQ
- パスに対する更新：RUQ

- これらが全部処理できるデータ構造・・・遅延SegmentTree

遅延セグ木のあれこれ

- 遅延セグ木でRAQとRUQと両方処理することってできるの？？

遅延セグ木のあれこれ

- 遅延セグ木でRAQとRUQと両方処理することってできるの？？

- できる！！
- 更新が来た時に下に更新が溜まってた：更新を上書き
- 更新が来た時に下に加算が溜まってた：更新で上書き
- 加算が来た時に下に更新が溜まってた：更新値に加算
- 加算が来た時に下に加算が溜まってた：加算値に加算
- 実装が大変かもしれませんが頑張ってください

名言紹介

- 個人的に好きな名言があります(誰の言葉か知りませんが)

名言紹介

- 個人的に好きな名言があります(誰の言葉か知りませんが)

木があるということは数列があるということ

典型知識

- 数列で解ける問題は木でも解けることが多い
- 部分木に対する処理が必要なときは

DFS木

を使う

  - このスライドではDFS木という言葉はDFSの訪問順で頂点番号を振りなおした木を意味する

DFS木

- 実装の詳細はきたむーさんのスライドを参照
- とりあえず[i,pos[i])がiを根とする部分木と一致する
- よって、部分木が直線になり、セグ木で普通に処理できる

典型知識

- 数列で解ける問題は木でも解けることが多い
- 今回はパスに対する処理が必要なときは

HL分解

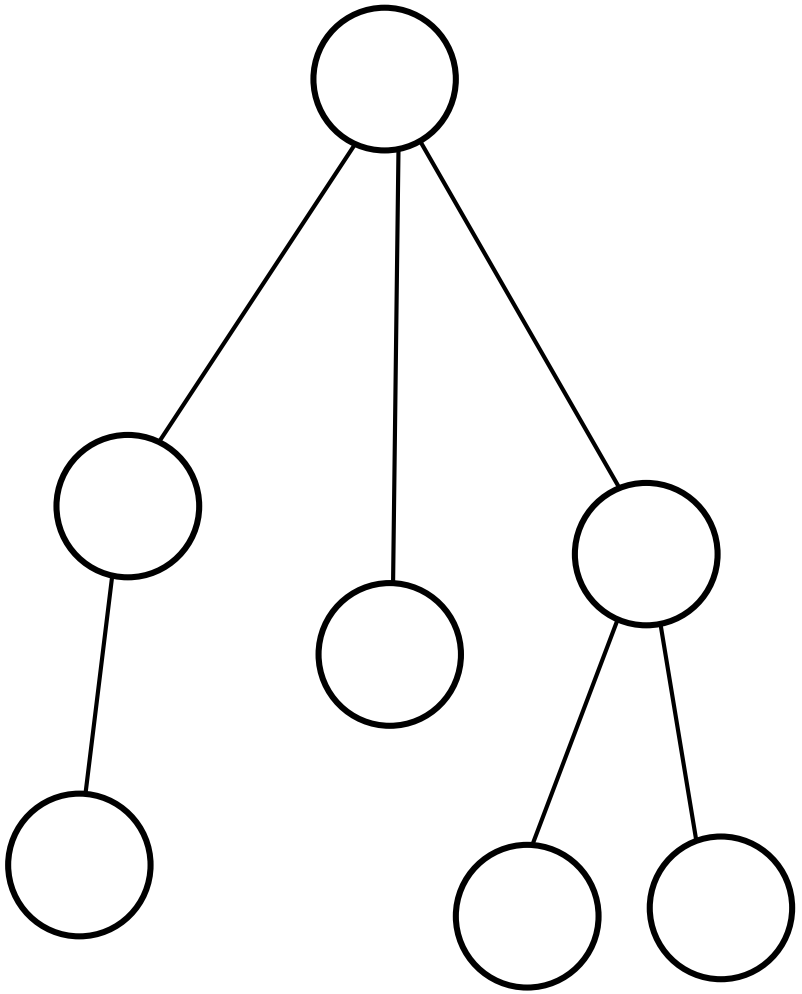
を使う

HL分解とは

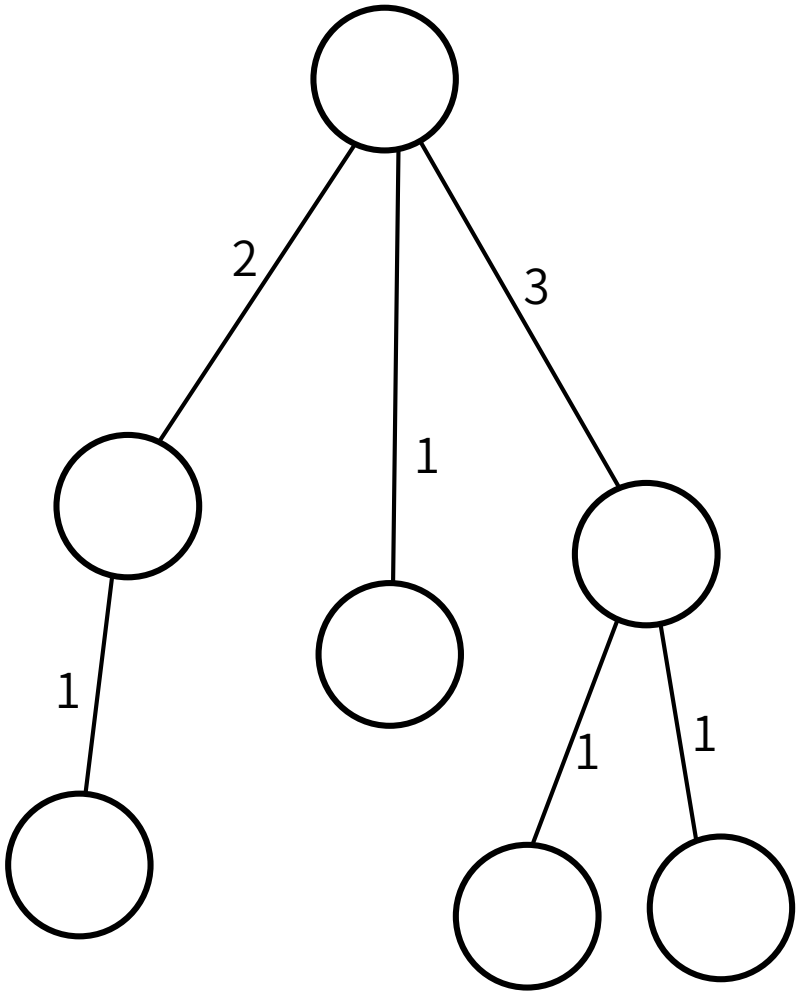
- 英語名：Heavy-Light Decomposition

1. 各辺にその下の頂点を根とした部分木のノード数を重みとして持たせる
2. DFSをして、ある頂点から出ている辺のうち、最も重いものをHeavy、それ以外をLightに設定する
3. Lightな辺を無視すると、直線がたくさん生じている
4. すべてのパスはこの直線を高々log N個通っている

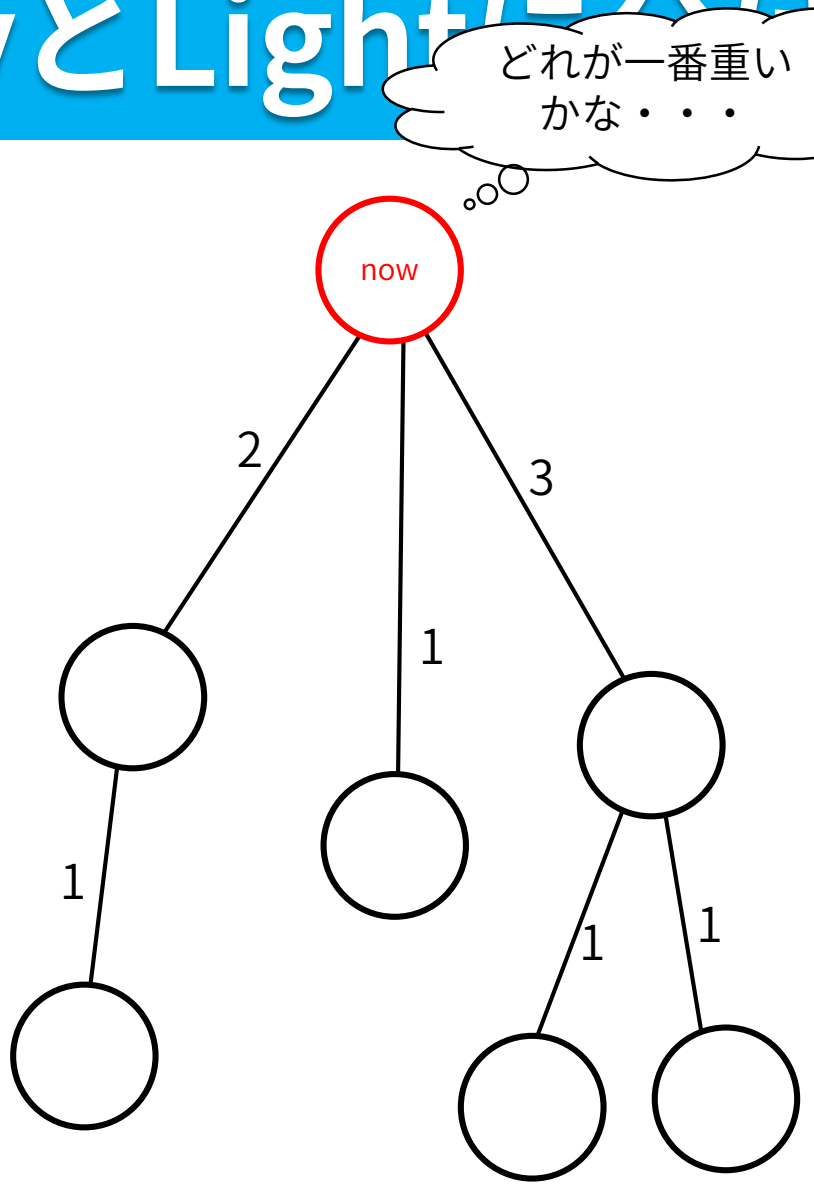
HL分解してみる



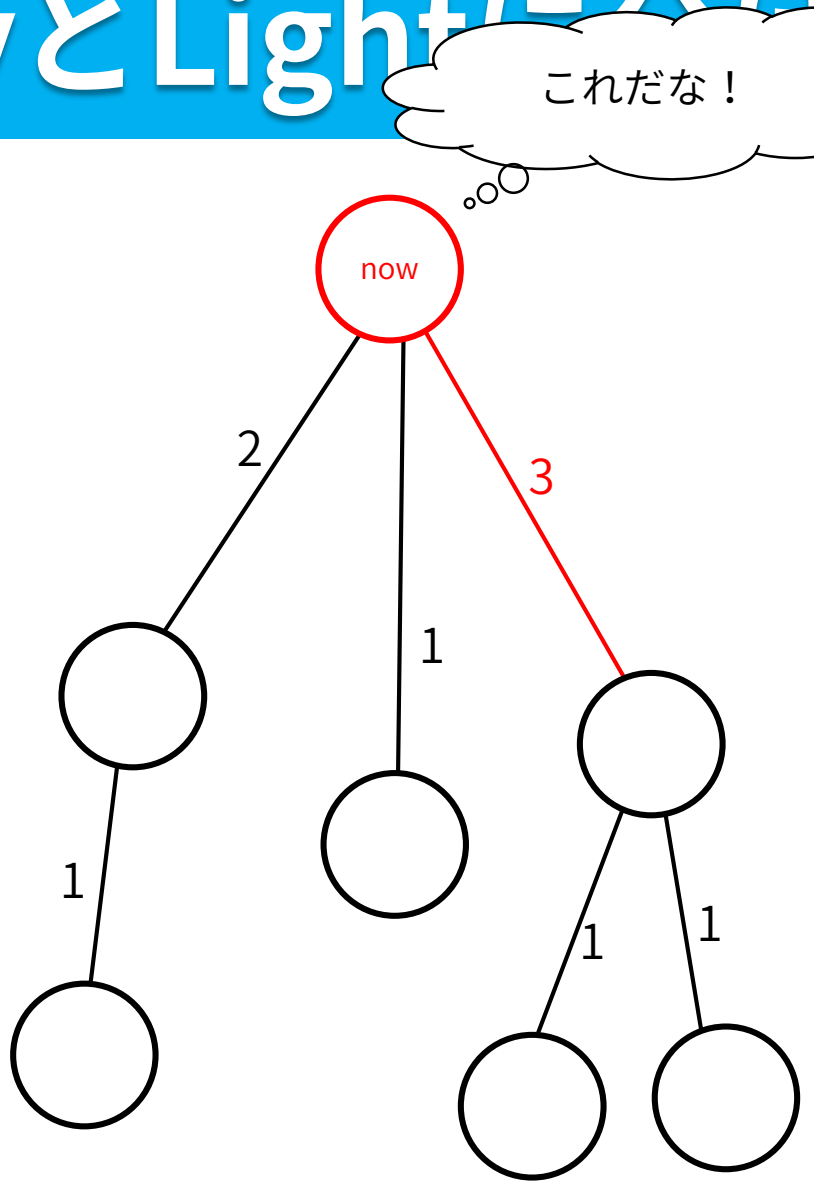
HL分解:部分木の大きさ求めてみた



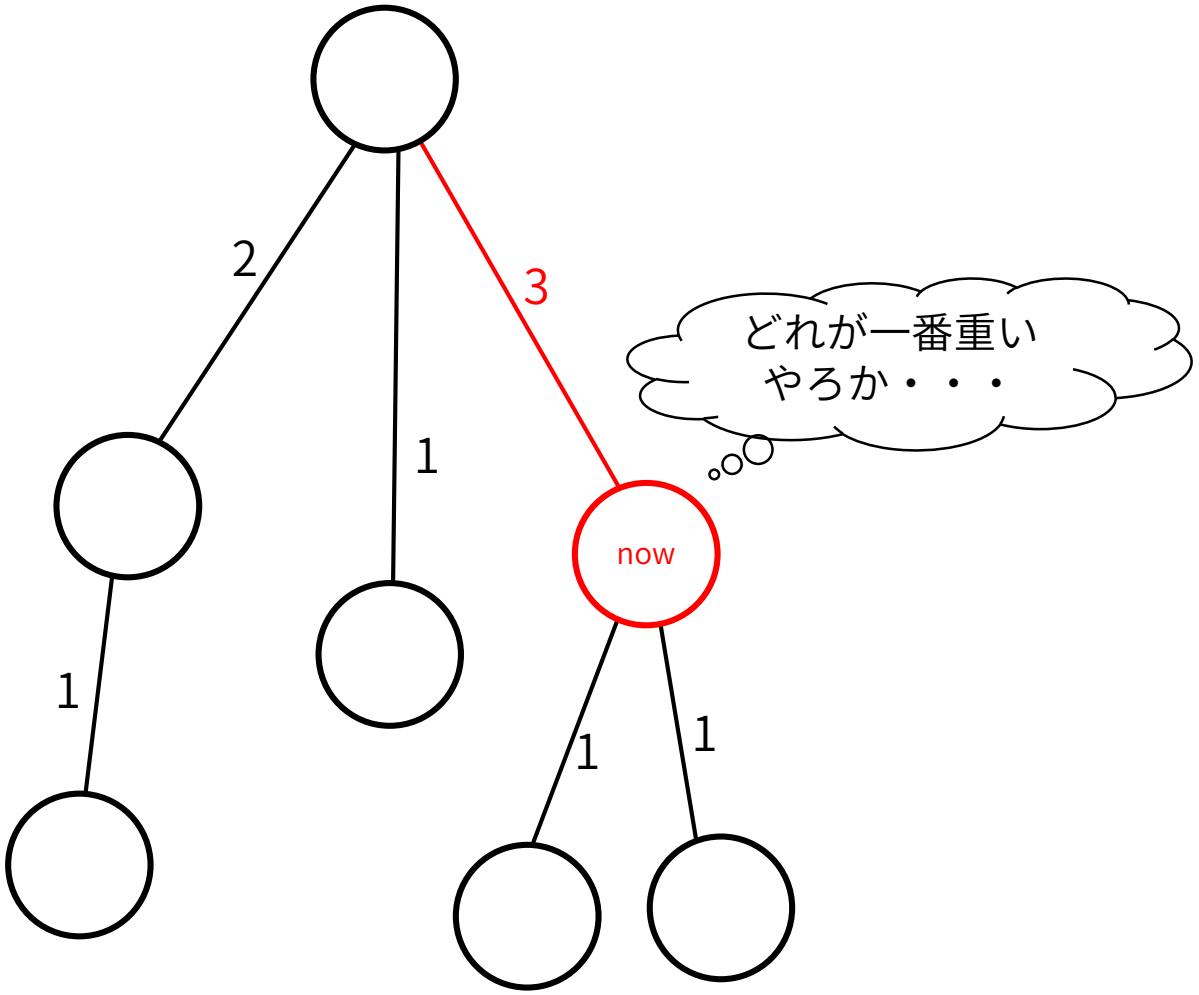
HL分解:HeavyとLightに分けてみた



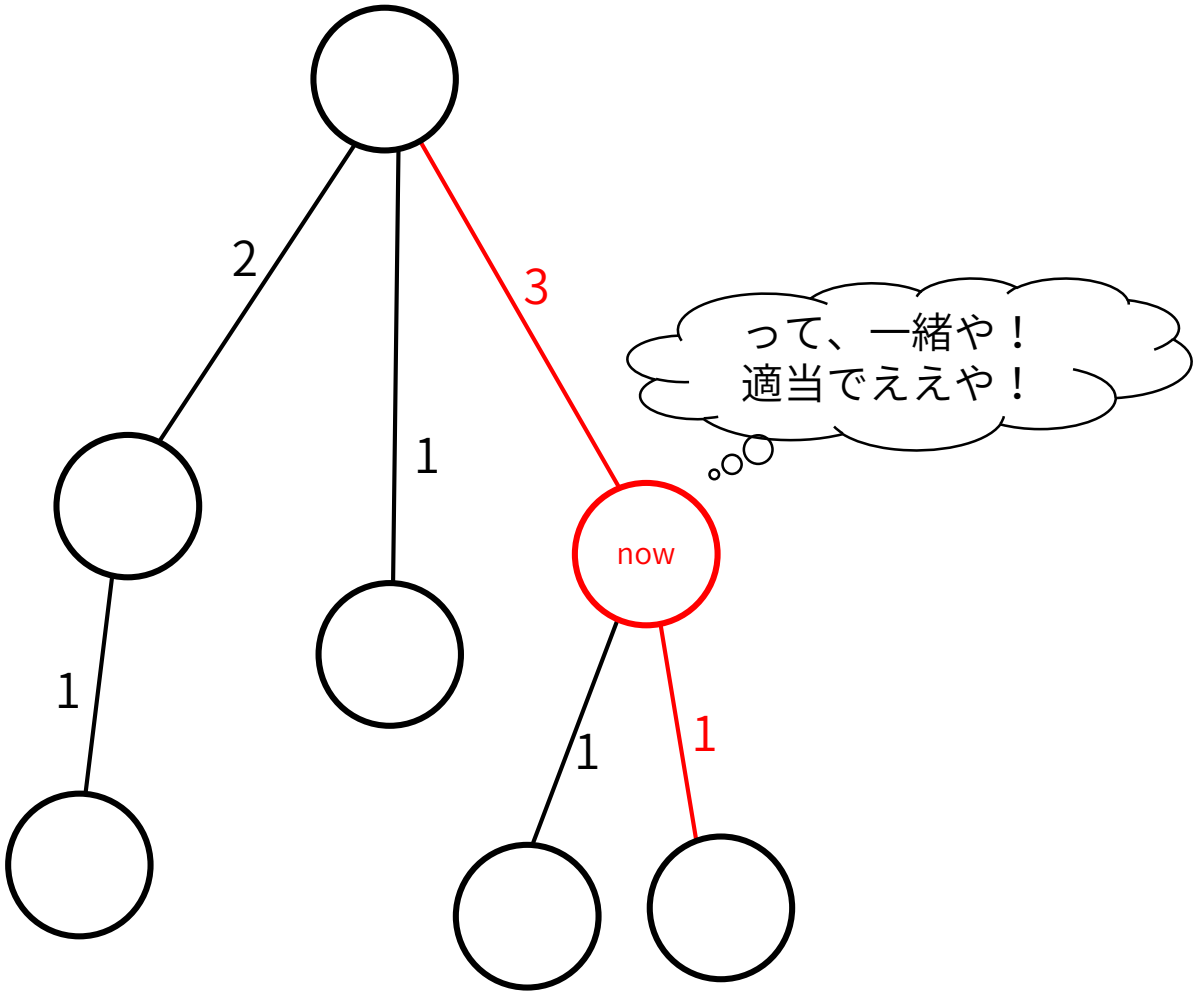
HL分解:HeavyとLightに分けてみた



HL分解:HeavyとLightに分けてみた

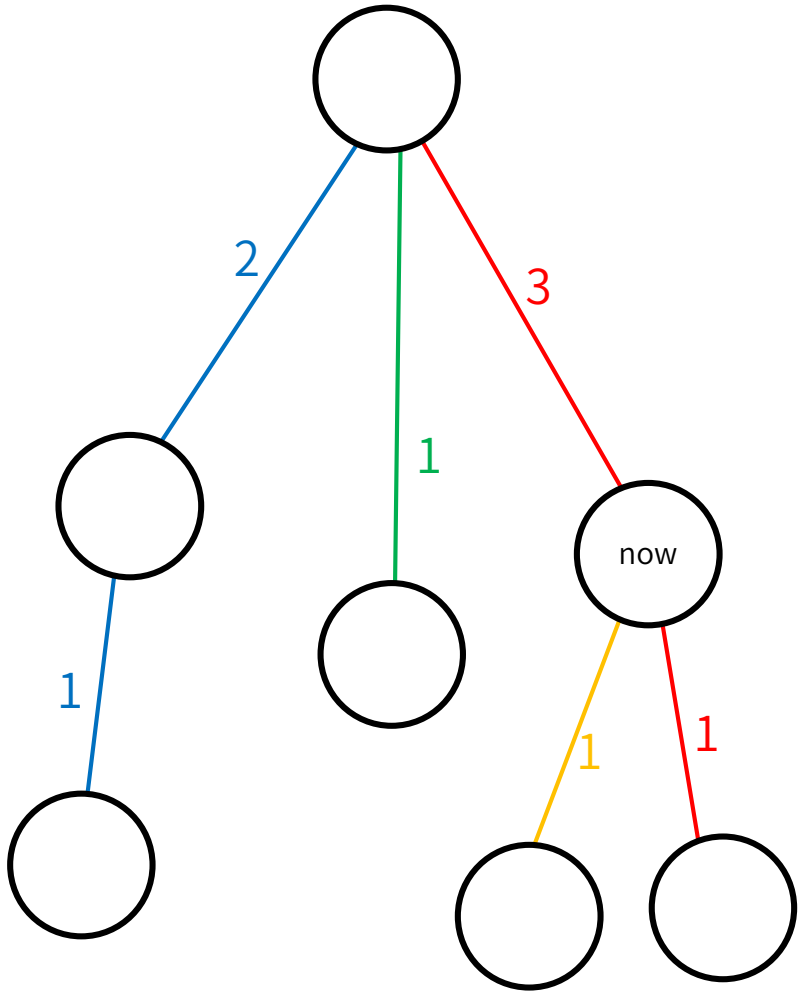


HL分解:HeavyとLightに分けてみた



HL分解:HeavyとLightに分けてみた

- というわけで、かなり飛ばしたが完成
- もっといい記事がたくさんあるのでそちらも併せて見てください



Thank you for reading!

ちょっと待った！！！

- 今回の問題では、クエリAが部分木クエリ、クエリBがパスクエリ、両方必要！！
- どうするねん！！！

実装面の話

- DFS木を作るときは当然DFSをする
- HL分解もDFSで実装可能

実装面の話

- DFS木を作るときは当然DFSをする
- HL分解もDFSで実装可能

- ということで、DFSをする関数を作って、DFS木を作りつつHL分解をすれば、DFS木とHL分解がリンクする！

実装面の話

- DFS木を作るときは当然DFSをする
- HL分解もDFSで実装可能

- ということで、DFSをする関数を作って、DFS木を作りつつHL分解をすれば、DFS木とHL分解がリンクする！
- つまり、部分木クエリとパスクエリが両方対応可能になる！

実装面の話

- DFS木を作るときは当然DFSをする
- HL分解もDFSで実装可能

- ということで、DFSをする関数を作って、DFS木を作りつつHL分解をすれば、DFS木とHL分解がリンクする！
- つまり、部分木クエリとパスクエリが両方対応可能になる！
- 天才か？？

DFS木を作りつつHL分解した結果

- DFS：$O(N)$
- クエリA(部分木クエリ)：$O(\log N)$
  - 数列に対してRAQは$O(\log N)$で処理できる
- クエリB(パスクエリ)：$O(\log^2 N)$
  - 各パスは高々$O(\log N)$個の分解された直線を通る
  - 数列に対してRUQ/RSQは$O(\log N)$で処理できる
- 全体では$O(N + Q \log^2 N)$

DFS木を作りつつHL分解した結果

- DFS：$O(N)$
- クエリA(部分木クエリ)：$O(\log N)$
  - 数列に対してRAQは$O(\log N)$で処理できる
- クエリB(パスクエリ)：$O(\log^2 N)$
  - 各パスは高々$O(\log N)$個の分解された直線を通る
  - 数列に対してRUQ/RSQは$O(\log N)$で処理できる
- 全体では$O(N + Q \log^2 N)$

- logが2つもついて大丈夫なの？
  大丈夫、実行時間制限を3秒にしといた。

  あまり変わらない気が・・・

サンプルコード

- さっきも書きましたがサンプルが250行もあってスライドには書けないので、こちらをご覧ください
- https://atcoder.jp/contests/iroha2019-day2/submissions/5202558
- 想定解は1.8秒ぐらいです
  - 本当はスクリプト言語の人とかに配慮したかったけど、O(QN)を落とすためにはこうせざるを得なかったのです
  - 定数倍が重くてTLEになっちゃった人はごめんなさい

ところで

- Typicalかそうでないかは簡単には判定できません
- 今回の問題も言ってしまえば

- RAQとRUQ両方に対応した遅延セグ木を書く
- DFS木を作りつつHL分解をする

- これだけなんですが、この知識が典型と呼べるかどうかはよくわかりません

Thank you for reading!