ビーバーの会合 (Meetings)

解説: 井上 航

小課題1(累積7点)

- すべての木を全探索します。
- たいへんですね
- 苦行す。
- あとこの先の話をすると、query(u,v,w)を聞きたいとき、おなじものが2つあるとそれを返す関数を作っておくと楽です

小課題2(累積17点)

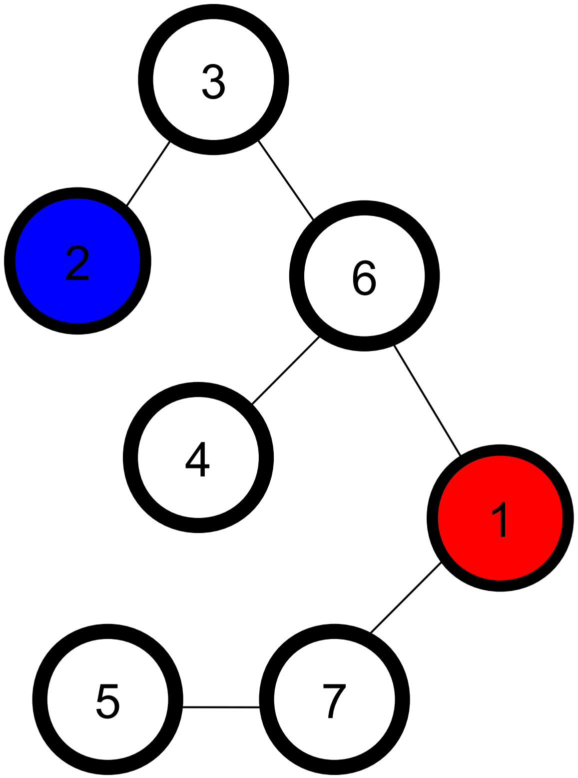
- n<=50です
- ありうるクエリは19600通りしかありません。
- なので、クエリをメモ化すればTLEしない限りとけます。
- 頂点u,vが直接つながれているかどうかを求めたいです。
- u,vが直接つながれている場合、すべてのwに対してquery(u,v,w)はuかvを返します
- 逆に、直接つながれてないならその間の場所が存在して、そこをwとするとquery(u,v,w)はu,v以外を返すことがあります
- よってすべてのu,vに対して求めてとけました

小課題3(累積31点)

- n<=300です
- やり方が2種類あります。ここでは、簡単な方を説明します。
- 頂点uについて、つながっている辺すべてを求めます。
- 「現在のところ、iに対して最も近い場所」をvとします。
- used[] という配列を持っておきます
- すべてのiに対して、query(u,v,i)を聞いて、それがuではないならqueryの答えの方がvに対して近づいています。v=query(u,v,i)にします. used[i]=true;
- クエリの答えがuならばuから見てiとvは別方向の頂点なので、後回しにします
- 最終的にはvはuに隣接する頂点です。bridgeを出力します。
- vを適当なusedでない頂点において、また同じことをします

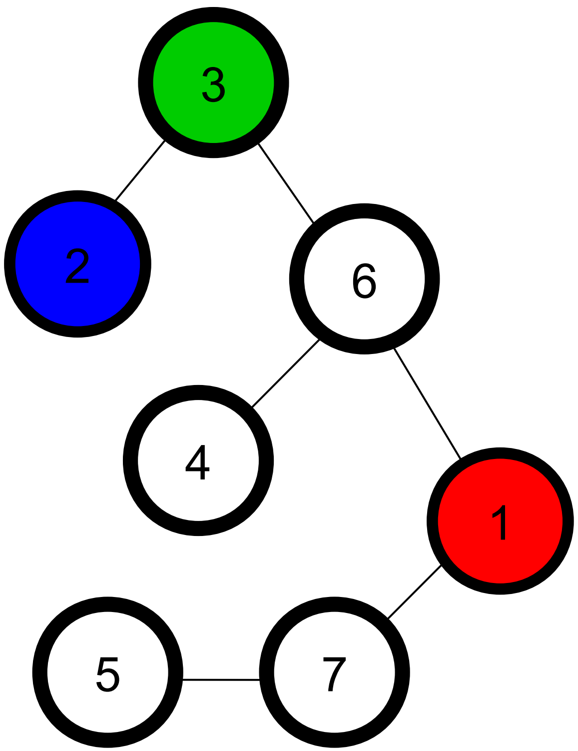
アルゴリズムの説明

- 頂点1につながる辺を求める。頂点1に近いのをとりあえず2にする

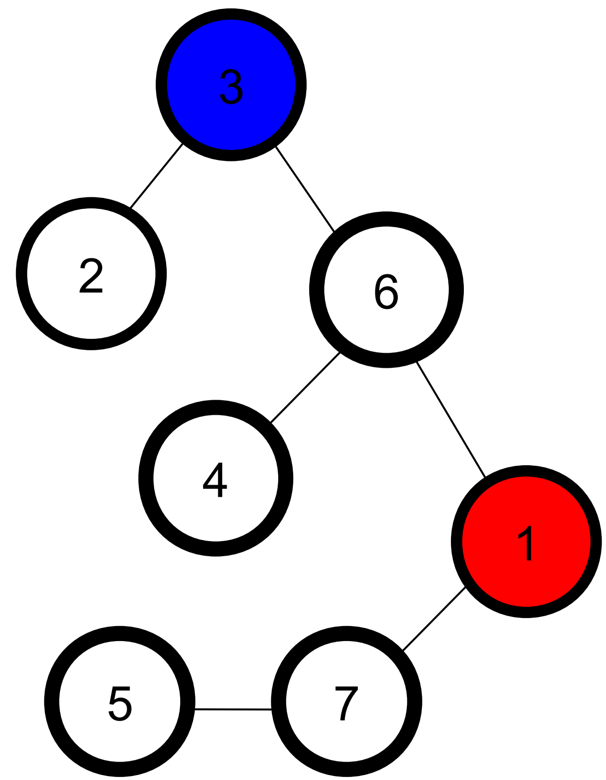


アルゴリズムの説明

- 頂点3について調べる

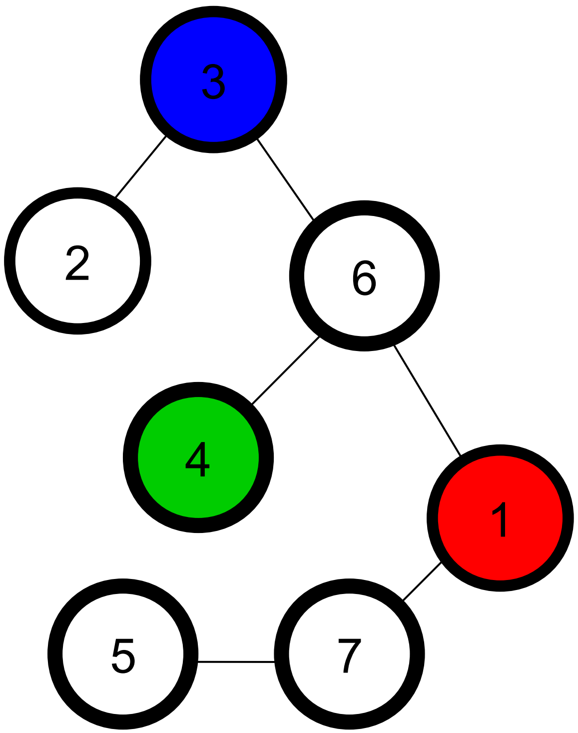


Query(1,2,3)=3
vが2から3に移動する

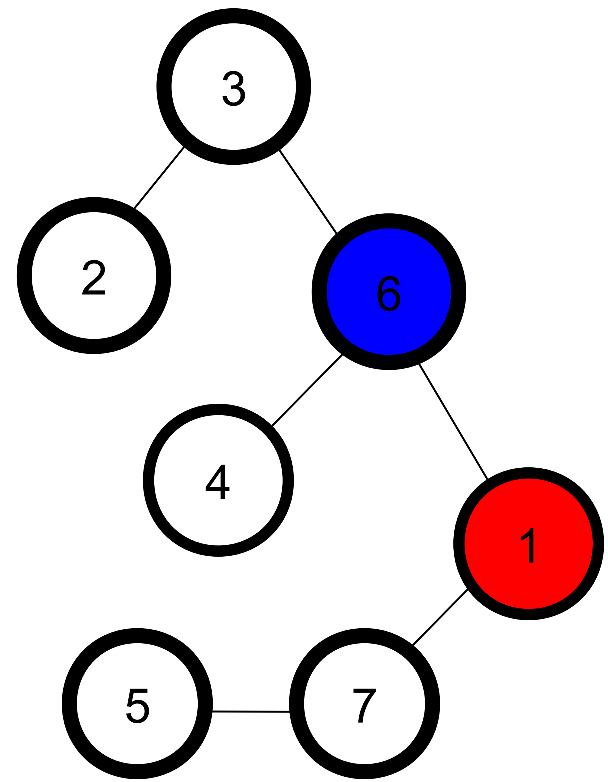


アルゴリズムの説明

- 頂点4について調べる

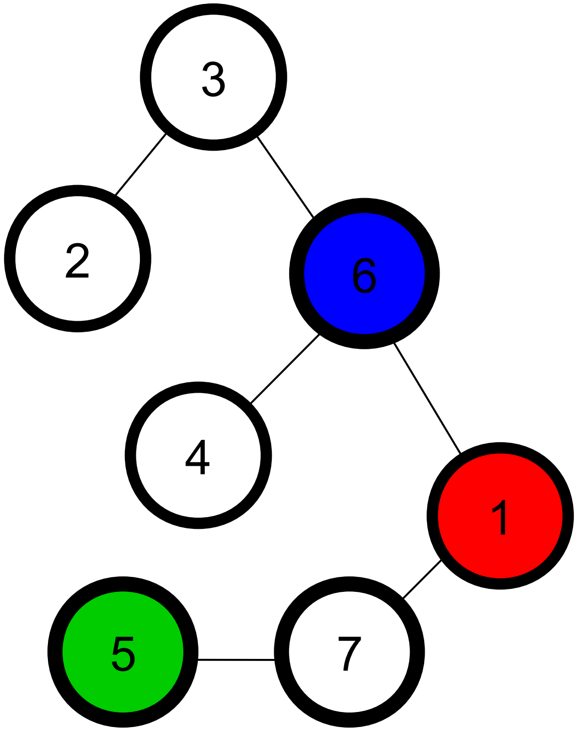


Query(1,3,4)=6
vが3から6に移動する

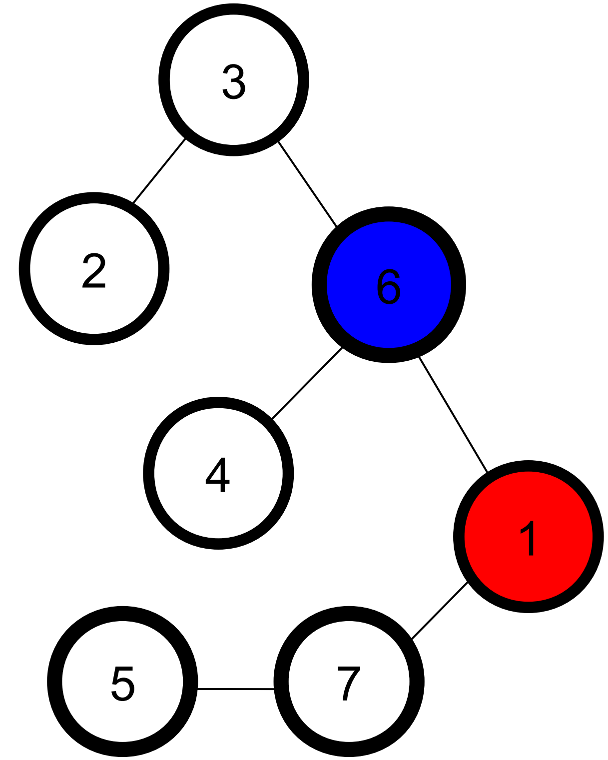


アルゴリズムの説明

- 頂点5について調べる

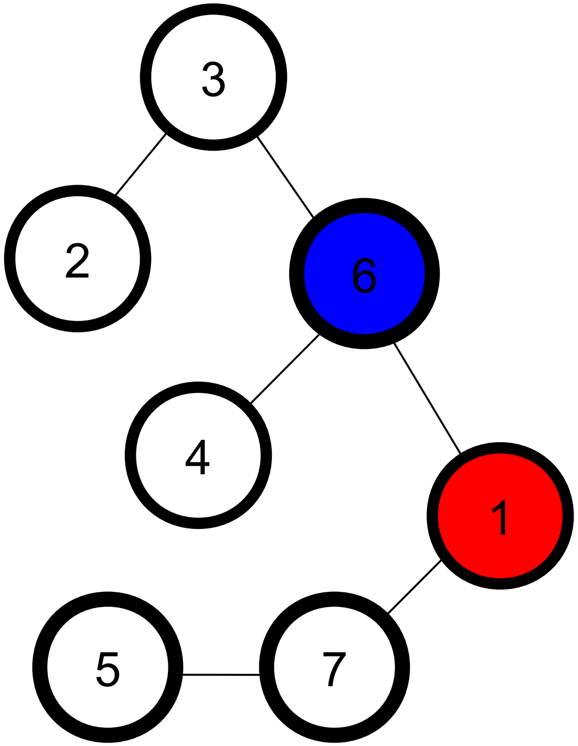


Query(1,6,5)=1
vはそのまま

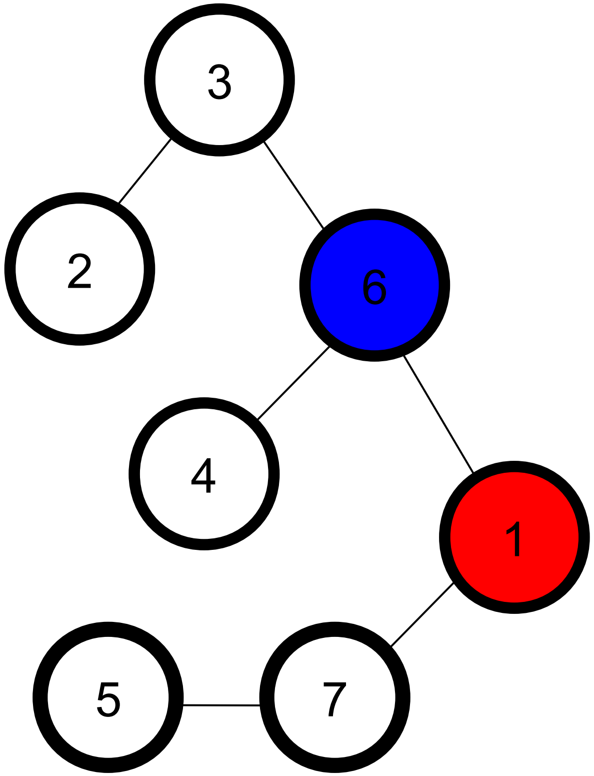


アルゴリズムの説明

- 頂点6について調べる



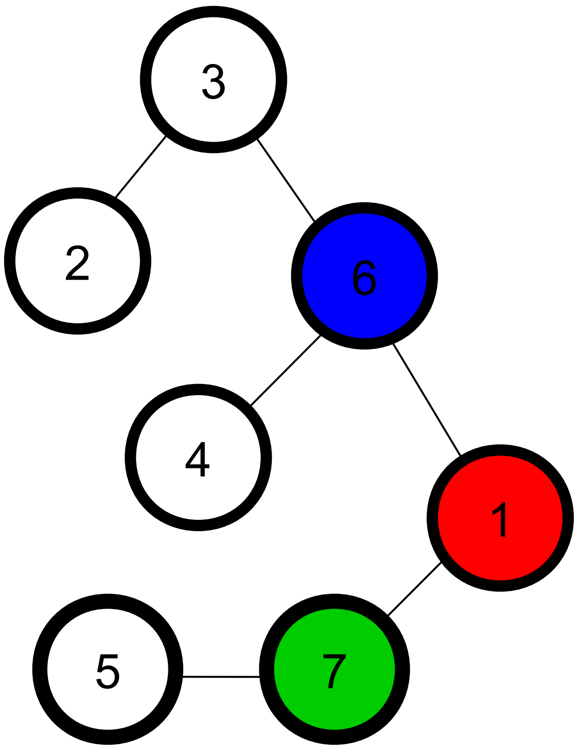
Query(1,6,6)=6
vは6

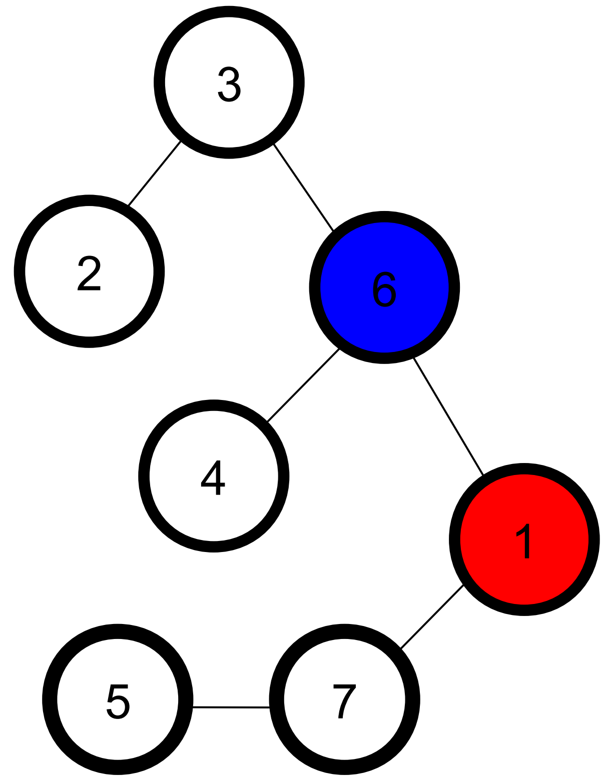


アルゴリズムの説明

- 頂点7について調べる

Query(1,6,7)=1
vはそのまま

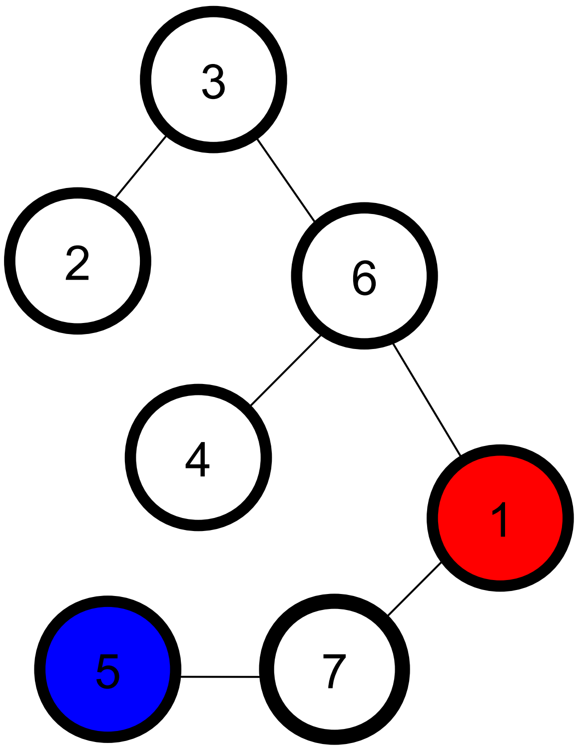




1-6の橋を見つけることができた

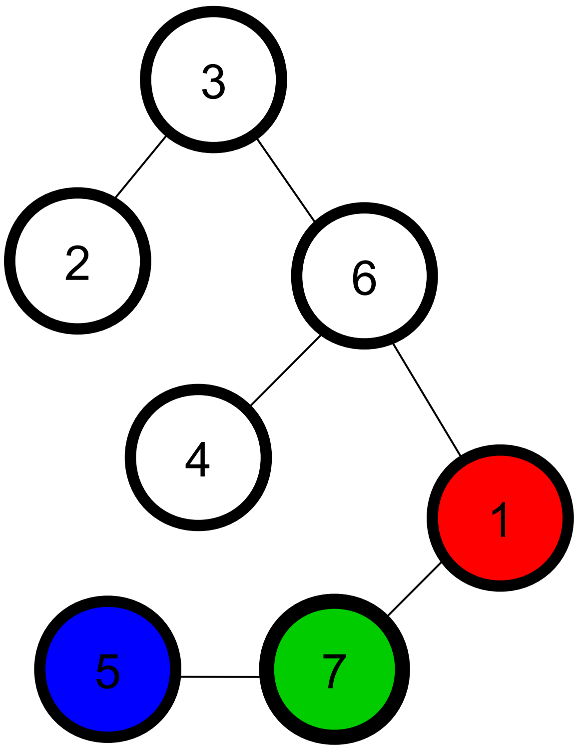
アルゴリズムの説明

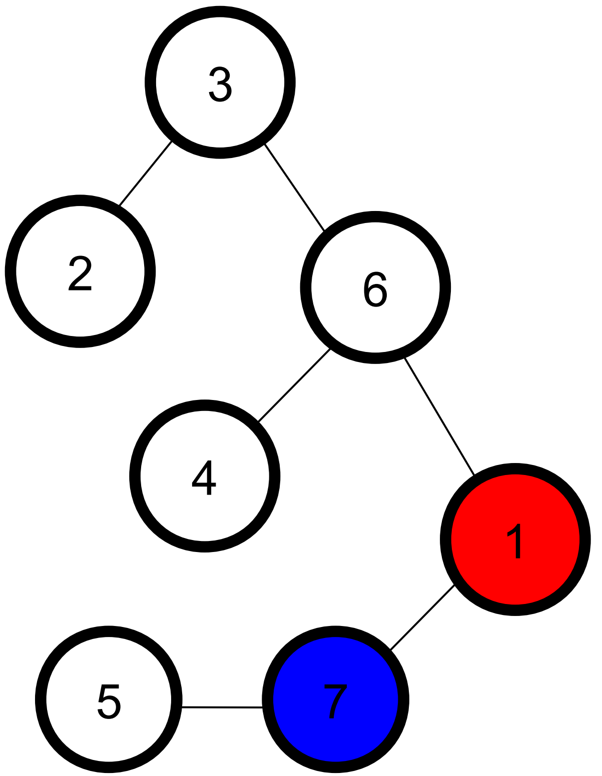
- 5と7が後回しになっているのでもう一周する、v=5



アルゴリズムの説明

- 頂点7について調べる





1-7の橋を見つけることができた

小課題3(続き)

- と思ったら解けていません。(最悪2*n^2回のクエリが必要)
- だから、枝刈りをします。
- bridgeを出力するときに、u<vのものだけ出力すればよいです。
- よって、まだ見つけてないvを見つけるために、for(i)の部分はu<iのものだけ調べればよいです。
- ただし、i<uのものを試してないので、最終的なvの方向に行く辺が、距離1になっているとは限りません。
- よって、距離1判定をするために、uと隣接することがすでに分かっている頂点すべてもiとしてqueryをして、距離1まで縮める必要があります。
- よって、今度こそ解けました。

小課題3(別解)

すでに木が存在したときに、それに対して新たな頂点がどのようなつながり方(間に入る、とか)をするかを考えます。

- 例：すでに1-2-3 の木があったとして、頂点4を考えます。query(2,3,4)は2でした。
- Query(1,2,4)が4なら木は1-4-2-3です。
- それが1なら木は4-1-2-3です。
- それが8なら木は1-8-2-3です。(このパターンに注意)

|
4

こっちの解法の良いところ

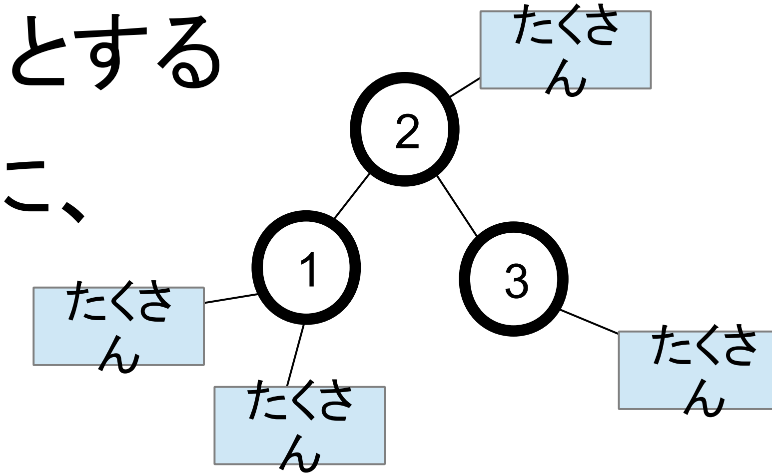
- 説明が楽
- クエリが少ない
- 満点につながる!!!

こっちの解法の悪いところ

- 実装がおもい

満点解法

- 小課題3の別解を改良します
- 重心分解をして、つながり方の頂点をうまく聞くと、とけます。
- 具体的に言うと、たとえば都市vがわかっていない状況で、query(1,3,v)を聞いたとする
- それが1ならvは1の部分木側に、
- 2ならvは2の部分木側に、
- 3ならvは3の部分木側に、
- それ以外なら1-2-3の木に対してvがついていることがわかる



満点解法

- この時、vの範囲が3分割されています。
- 「こっちにある」の方向に対して、再帰的に求めます。
- クエリを聞くときに、最悪のケースでの残った頂点の個数を最小にするように聞くとよいです。
- 具体的には、query(v,(重心から延びる1番めに部分木が大きい方の頂点),(重心から延びる2番めに部分木が大きい方の頂点))と聞くなどがあります。
- この方針だと、たぶん100点を得ます。

4万回以内の証明(かなり難しいです)

n 頂点からなる木がわかっていて 1 頂点追加する場合のコスト

追加したい頂点を q とする.

重心分解する. 重心を g とする.

g の隣接頂点を v1, v2, ..., vd として, g を取り除いたときの部分木で vi を含むものの大きさを si とする (s1 >= ... >= sd). その部分木を Si とおく.

q, vi, vj でクエリを投げると，答えは次のどれか：

- vi または vj: Si か Sj の中.
- g: Si, Sj の中にはない.
- それ以外: g と (vi か vj のどっちか) の間．もう 1 回クエリを投げると完全にわかる.

最後の「もう 1 回」は 1 回だけしか起きないので無視しておく.

まず d が偶数の場合だけ考えると，次のどれかが起きることがわかる:

- 1 回のクエリを使って，大きさが高々 s1 の部分木に範囲を絞る.
- 2 回のクエリを使って，大きさが高々 s3 の部分木に範囲を絞る.
- ...
- 9 回のクエリを使って，大きさが高々 s17 の部分木に範囲を絞る.

d が奇数の場合は，次のどれかも起きうる：(部分木が 1 個になった地点で重心分解をやり直す)

- 1 回のクエリを使って，大きさが高々 s3+1 の部分木に範囲を絞る.

- 2 回のクエリを使って，大きさが高々 s5+1 の部分木に範囲を絞る.

-...

- 8 回のクエリを使って，大きさが高々 s17+1 の部分木に範囲を絞る.

ところで，頂点数が 1 増えただけでクエリ回数が 2 回以上増えるわけはないので，この奇数の場合は無視していいことがわかる.

（例えば「8 回で s17+1」より「9 回で s17」のほうがコストは大きい）

これをパソコンで計算すると，2 頂点，3 頂点，…，1999 頂点についてのこのコストの和は（「もう 1 回」を含めて） 39632 になることがわかる.

(s1 <= n/2，sk <= n/k に注意)

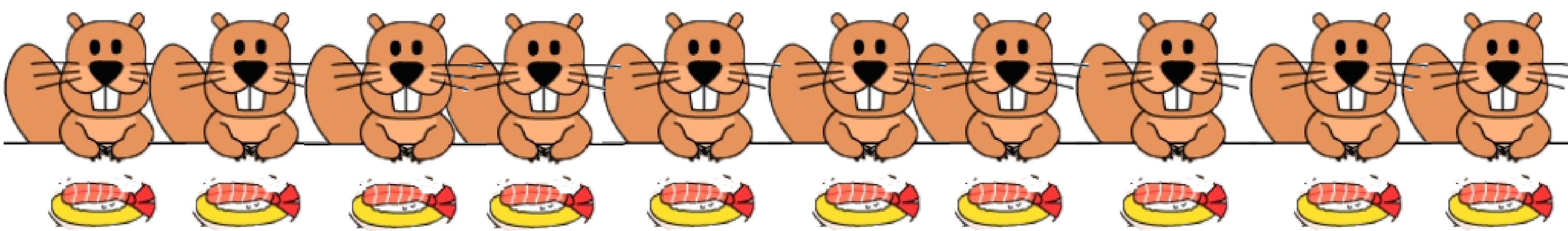
実際には見る点の順番をrandomにすればかなりうまくいきます

得点分布

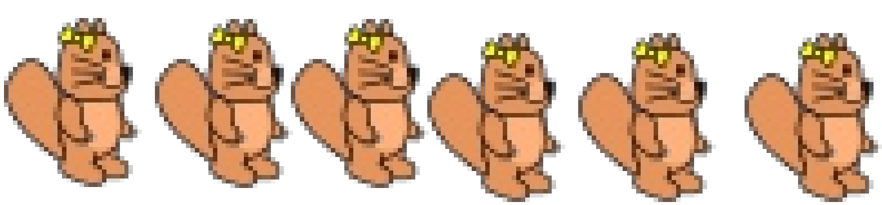
100

78

29



17



0