Resolução Contest 2: Jakarta 2018

André Amaral de Sousa

Autor

Sobre

- Ex-Maratonista (ICPC 2013)
- Ex-Olímpico (IOI 2012)
- Problem Setter na Brasileira 2019 e 2020
- Engenheiro de Computação (POLI-USP)
- Professor no colégio ETAPA
- Contato:
  - andremfq@gmail.com

Placar

Begin: 2022-01-28 08:00 UTC-3

Brazilian ICPC Summer School Level B 28/01

End: 2022-01-28 13:00 UTC-3

Ended

Overview | Problem | Status | Rank (05:00:00) | 0 Comments

Setting | ☆Favorite | Clone

Rank	Team	Score	Penalty	A 25 / 130	B 0 / 0	C 0 / 0	D 23 / 107	E 0 / 1	F 3 / 19	G 5 / 57	H 3 / 6	I 44 / 51	J 7 / 31	K 0 / 6	L 28 / 126
1	ufcg_misturados (ufcg...	8	1524	02:17:22 (-5)			00:58:48 (-1)		04:57:34 (-2)	03:53:34 (-3)	04:22:32 (-1)	00:06:11	03:35:18 (-2)		00:33:11
2	thiagob (thiagob)	7	1277	01:21:45 (-3)			02:28:29 (-3)			04:58:34 (-5)	03:44:10	00:07:18	03:12:46 (-4)		00:24:56
3	Carlos_Cabral (Carlos_...	6	1002	01:01:38			03:09:51 (-3)		02:37:26 (-7)	04:36:26 (-2)		00:24:26			00:53:11
4	sqlados (SQLados)	6	1054	01:05:37			02:43:41 (-2)				04:23:17 (-1)	00:08:46	03:02:57 (-2)		02:10:19 (-7)
5	pvtdacomp (time sem ...	5	428	00:46:44 (-1)			01:44:20 (-3)			(-6)		00:06:41	02:21:02		00:49:22
6	Artur_Gaspar (Artur_G...	5	437	00:40:47 (-1)			01:45:43			03:43:34		00:07:12			00:40:08
7	Balloonatics (Machado...	5	547	00:36:59 (-3)			01:23:00 (-4)			(-2)		00:04:48	02:43:20		00:39:21 (-4)
8	Natan (NatanSG)	5	669	01:03:31			02:53:37 (-6)					00:10:06	03:48:37 (-1)	(-5)	00:53:09
9	naniim (naniim)	5	673	00:39:40 (-3)			01:44:57 (-2)			03:42:45 (-8)	(-1)	00:04:30			00:41:24
10	Reim (Daniel Hosomi)	5	710	00:37:56			01:41:51 (-1)					00:10:42	04:40:29 (-9)		00:59:38 (-1)
11	GuilhermeCpp (Guilher...	5	892	01:12:15 (-1)			02:17:02 (-6)		04:24:23 (-7)			00:16:20 (-1)	(-2)		01:02:22 (-2)
12	batleite (batleite)	4	414	01:01:46 (-2)			01:45:48 (-2)			(-3)		00:21:26	(-1)		01:45:21 (-2)
13	2Victors_e_1PNC (2 Vic...	4	427	01:26:25 (-2)			02:10:20 (-3)					00:07:08			00:43:53 (-3)
14	O_confia (emn2 & gab...	4	503	01:59:01 (-2)			02:51:09 (-4)	(-1)		(-6)		00:33:41	(-2)		01:00:05

Comentários Gerais:

- Contest disponível no GYM
- 5h, 4 estrelas
  - Muitos contests 4 estrelas estão num nível alto pra Maratona Brasileira (quem vai brigar pelos primeiros lugares)
  - Se sobrar MUITOS problemas para upsolvar, significa que está muito pesado, se for o caso foque nos 3 estrelas.
- Tutorial
  - Não falarei de todos os problemas
- É muito importante upsolvar os problemas
  - Pense nos problemas que não teve tempo durante o contest
  - Se mesmo após pensar não conseguir resolver, leia o tutorial e resolva
- Vou resolver apenas os problemas F, G, H, K

Ideia

- Podemos facilmente calcular a quantidade de Balões que o A vai conseguir, digamos Qtd. E perceba também que os momentos onde A recebe balões já estão determinados
- Só precisamos verificar se é possível fazer B resolver no máximo Qtd - 1, note que não queremos minimizar a quantidade de problemas de B
- Outra observação é que se A vai estourar um balão, SEMPRE compensa estourar em um momento onde B iria resolver um problema, assim faz B perder mais tempo
- Suponha que em um dado momento A tem balão pra estourar e B vai completar seu i-ésimo problema (B[i]), como saber se vale a pena A estourar esse balão ou esperar pra estourar em algum momento no Futuro ?
- Observação Principal: só vale a pena A estourar o balão se B[i] = máximo(B[i], B[i + 1], ..., B[Qtd]). Pois suponha que B[i] não seja o máximo, ou seja, existe um j > i (e j <= Qtd) tal que B[j] > B[i]. Então vale mais a pena esperar pra estourar esse balão quando B for terminar o problema j, assim esse balão fará ele perder B[j] de tempo. Note que não há garantia que B chegará a resolver o problema j, mas se ele não chegar a resolver, encontramos uma solução!

Ideia

- Observação Principal: só vale a pena A estourar o balão se B[i] = máximo(B[i], B[i + 1], ..., B[Qtd]). Pois suponha que B[i] não seja o máximo, ou seja, existe um $j > i$ (e $j <= Qtd$) tal que $B[j] > B[i]$. Então vale mais a pena esperar pra estourar esse balão quando B for terminar o problema j, assim esse balão fará ele perder B[j] de tempo. Note que não há garantia que B chegará a resolver o problema j, mas se ele não chegar a resolver, encontramos uma solução!
- Perceba também que maximizar a escolhe do balão atual não compromete a escolha dos balões futuros,

F - Popping Balloons

Ideia

- Como implementar essa ideia Gulosa ?
- Precompute Qtd e o máximo de sufixo pro B (só até o índice Qtd)
- Há varios eventos acontecendo, mas eles são de dois tipos, ou o A resolve mais um problema, ou o B "resolve" mais um problema (mas o A pode estourar um balão nesse momento).
- Para processar os eventos, mantenha:
  - Próximo tempo que cada um dos competidores irá resolver mais um problema
  - Índice do problema que cada competidor está
  - Quantidade de Balões do A
- Enquanto o próximo tempo de algum dos competidores for menor ou igual a m, iremos processar o evento:
  - Se o menor próximo tempo a resolver um problema for do A, então incrementamos a quantidade de balões, e atualizamos seu próximo tempo e o índice do seu próximo balão
  - Se for o B, precisamos verificar se o A irá estourar um balão nesse momento ou não. Basta verificar se o balão atual é o máximo do sufixo correspondente, e se o A tem algum balão.
  - Se sim, então decrementamos o número de balões de A, atualizamos o próximo tempo que o B irá resolver um balão, MAS note que o índice do balão continua o MESMO.

Ideia

- Enquanto o próximo tempo de algum dos competidores for menor ou igual a m, iremos processar o evento:
  - Se o menor próximo tempo a resolver um problema for do A, então incrementamos a quantidade de balões, e atualizamos seu próximo tempo e o índice do seu próximo balão
  - Se for o B, precisamos verificar se o A irá estourar um balão nesse momento ou não. Basta verificar se o balão atual é o máximo do sufixo correspondente, e se o A tem algum balão.
  - Se sim, então decrementamos o número de balões de A, atualizamos o próximo tempo que o B irá resolver um balão, MAS note que o índice do balão continua o MESMO. Já aproveite e vá construindo a resposta, ou seja, adicione esse tempo à um vector resposta
  - Se não, então incrementamos a quantidade de balões de B e atualizamos seu próximo tempo e o índice do seu próximo balão. Perceba que se B chegar a resolver seu balão de índice Qtd, então a resposta é impossível
- Complexidade: O(N) Pois o A só irá alterar seu tempo no máximo O(N) vezes, e o B apesar de poder continuar no mesmo índice, cada vez que isso acontece ele usa um balão de A, o que só acontece no máximo N vezes, logo só altera seu tempo no máximo O(N) vezes
- Tome cuidado com a implementação!

G - Go Make It Complete

Ideia

- Queremos o maior k que exista uma ordem possível de construir as arestas. É natural pensar em Busca Binária no k.
- Perceba que fixado um k, quando uma aresta se torna válida (soma dos graus >= k) ela já pode ser criada a qualquer momento dali pra frente, pois os graus só aumentam.
- Outra observação importante é que se em um dado momento várias arestas estão válidas, você pode criá-las em qualquer ordem, ou seja, a ordem relativa entre elas não é importante.
- Dessa forma, para testar se para um dado k existe resposta, podemos manter todas as arestas que faltam ordenadas pela soma dos graus em alguma estrutura, e a cada passo podemos pegar a que possui maior soma dos graus e criá-la. Se em algum momento a soma dos graus de alguma aresta for < k, então este k não é válido, senão será.
- Detalhe ao criar uma aresta, temos que tirá-la da estrutura com as arestas, incrementar os graus de seus vértices, e recalcular a soma dos graus de toda aresta que ainda falte criar e que algum dos nós seja um desses incrementados (e atualizar a estrutura com as arestas)

G - Go Make It Complete

Ideia

- Uma forma de realizar essas operações é mantendo um Set com as arestas, e uma matriz de adjacências marcando quais arestas já foram criadas.
    - Com o Set conseguimos a maior aresta ainda não criada
    - Apagamos ela do Set, marcamos essa aresta como criada, e incrementamos os graus dos dois vértices
    - Para cada vértice, passamos por todos os outros vértices, e se a aresta ainda não foi criada, atualizamos seu valor no Set (apagamos o valor antigo e inserimos um com a soma de graus nova)
- Mas qual a complexidade dessa solução ?
- O(logN) passos na Busca Binária
- $O(N^2)$ arestas para serem criadas
- O(NlogN) para processar cada aresta, O(N) para checar os outros vértices, e O(logN) cada operação no Set
- Portanto: $O(N^3 \log^2 N)$ o que dá TLE
- Pelos limites, que estão apertados, vemos que precisamos buscar uma ideia $O(N^3)$

Ideia

- Como podemos nos livrar da Busca Binária ?
- Observe que a ordem com que estamos criando as arestas INDEPENDE do k da busca binária, ou seja, todo passo da Busca Binária estamos fazendo a mesma coisa novamente.
- Portanto não precisamos da Busca Binária, podemos fazer a mesma coisa e apenas guardar qual foi o menor valor que apareceu como soma de graus
- Outra forma de enxergar é a seguinte, suponha a aresta que atualmente possui a maior soma de graus, o valor de k não pode ser maior que esse valor pois senão daqui pra frente nenhuma aresta seria criada e o processo estaria parado.
- Portanto o valor de k precisa ser menor ou igual ao valor da aresta de maior valor. Conforme já observamos, se uma aresta é válida ela pode ser criada a qualquer momento (a ordem entre arestas já válidas não importa) logo podemos a cada passo criar a aresta de maior valor.

G - Go Make It Complete

Ideia

- Como podemos nos livrar do Set ?
- Perceba que o valor de uma aresta é de 2 à 2*(N-1), pois é a soma de dois graus, ou seja, há apenas O(2N) valores que as arestas podem assumir
- Podemos criar uma lista de arestas (um vector com os índices das arestas, ou um vector de pair com os dois vértices das arestas) para cada possível valor e colocar todas as arestas com um dado valor naquela lista
- Ao invés de pegar a maior aresta a cada passo, podemos passar pelos possíveis valores do maior ao menor (de 2*(N - 1) à 2) e ir criando todas as arestas naquela lista
- Ao criar uma aresta precisamos:
    - Arrancá-la da lista, podemos por exemplo processar o vector de trás pra frente, assim remover fica O(1)
    - Marcar que essa aresta está criada
    - Incrementar os graus dos dois vértices
    - Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES

G - Go Make It Complete

Ideia

- Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES
- Por exemplo, vamos supor que a aresta (3, 8) estava com valor 5 e foi pro valor 6

Listas: **1** **2** **...** **5** **6** **...** **2*(N-1)**

Lista 5:

...	...
3	8
...	...
x	y

Lista 6:

...	...
u	v

- Como podemos identificar onde está a aresta (3, 8) na lista de valor 5 ? Como arrancar ela de lá e passar pra lista de valor 6 ?

G - Go Make It Complete

Ideia

- Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES
- Por exemplo, vamos supor que a aresta (3, 8) estava com valor 5 e foi pro valor 6

Listas: 1 2 ... 5 6 ... 2*(N-1)

Lista 5:

…	…
3	8
…	…
x	y

(id[3][8] marca a linha 3, 8)

Lista 6:

…	…
u	v

- Podemos manter o índice de cada aresta em uma matriz de marcação, por ex id[3][8]

G - Go Make It Complete

Ideia

- Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES
- Por exemplo, vamos supor que a aresta (3, 8) estava com valor 5 e foi pro valor 6

- Como a ordem entre as arestas de mesmo valor não importa, podemos trocar essa aresta com a última

G - Go Make It Complete

Ideia

- Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES
- Por exemplo, vamos supor que a aresta (3, 8) estava com valor 5 e foi pro valor 6

Listas: **1** **2** **…** **5** **6** **…** **2*(N-1)**

Lista 5:

	…	…
	x	y
	…	…
id[3][8]	3	8

Lista 6:

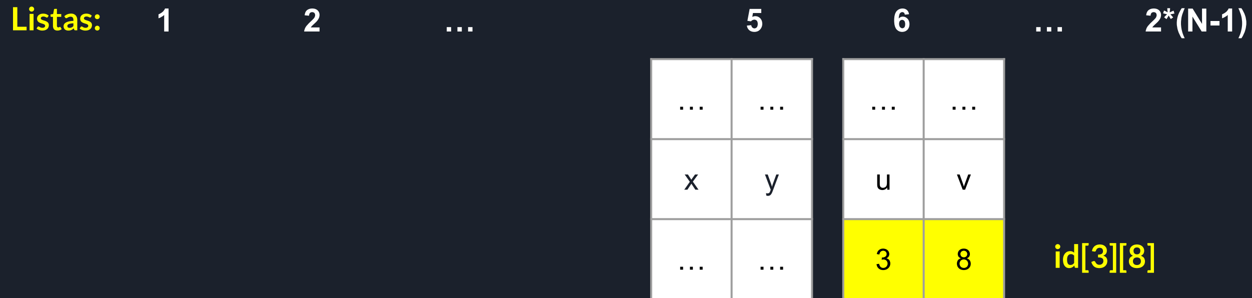
…	…
u	v

- Como a ordem entre as arestas de mesmo valor não importa, podemos trocar essa aresta com a última. Agora basta arrancar a última (pop_back() do vector é O(1)).

G - Go Make It Complete

Ideia

- Atualizar o valor de toda aresta não criada desses dois vértices -> ISSO NÃO É TÃO SIMPLES
- Por exemplo, vamos supor que a aresta (3, 8) estava com valor 5 e foi pro valor 6



- E então inserir na lista de valor 6, cuidado para manter id[3][8] com o índice correto

Ideia

- Detalhe se estivermos processando as arestas de valor 6 por exemplo e durante esse processo uma de valor 7 é gerada, como proceder ?
- Se estamos criando as arestas de valor 6 então k será no máximo 6 (pode ser que seja menor), e já criamos todas com valores maiores
- Note que é impossível gerar uma aresta de valor 8 ou maior, pois quando atualizamos o valor de uma aresta não criada, seu valor apenas soma 1
- Portanto ao criar uma aresta de valor 6 o máximo que pode acontecer é gerar uma aresta de peso 7, podemos proceder de duas formas:
    - Já criar ela imediatamente pois sabemos que a resposta é no máximo 6
    - Não fazer nada, deixa ela na lista de valor 6 mesmo (ainda que esteja errada) pois ela será criada nos próximos passos.

Complexidade Final: $O(N^3)$

H - Lexical Sign Sequence

Ideia

- Como as restrições são do seguinte tipo: a soma de um determinado intervalo precisa ser PELO MENOS um determinado valor, então para saber se existe solução, basta colocar 1 em todas as posições ainda não determinadas e testar, pois como estamos escolhendo o valor máximo onde podemos, se existir solução, esta será uma solução.
- Em um vetor auxiliar, colocamos 1 em todas as posições não determinadas (se já estiver determinada apenas copia), depois usando Somas Parciais podemos verificar se cada uma das restrições é satisfeita ou não.
- E já aproveitamos para calcular para cada restrição, quanto excedeu do valor que era desejado (usaremos na solução)

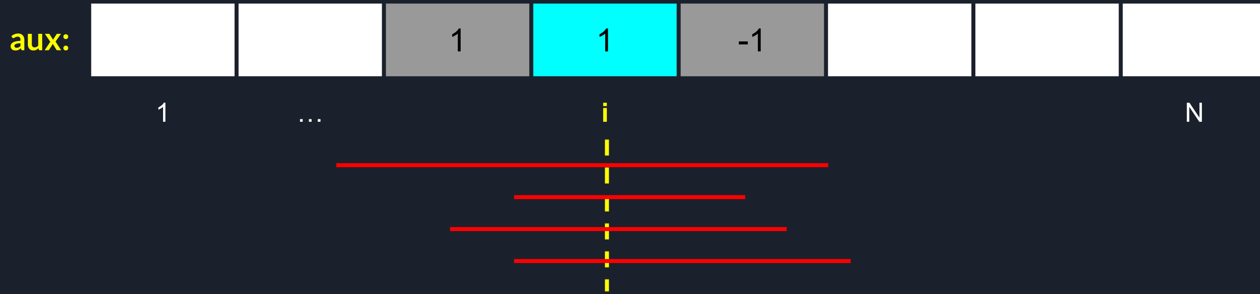
H - Lexical Sign Sequence

Ideia

- Agora que sabemos que há solução (pois se não existir basta imprimir impossível), iremos determinar a menor solução lexicograficamente termo a termo de 1 à N.
- A ideia é simples, para determinar o valor da posição i:
    - Se é uma posição já determinada não temos escolha, use o valor determinado
    - Senão já sabemos que existe uma solução com 1 nessa posição (solução atual), mas queremos verificar se é possível trocar esse 1 por -1.
    - Se for possível, colocamos -1 pois sempre vale a pena lexicograficamente
    - Senão, deixamos como 1 e seguimos para a próxima posição

H - Lexical Sign Sequence

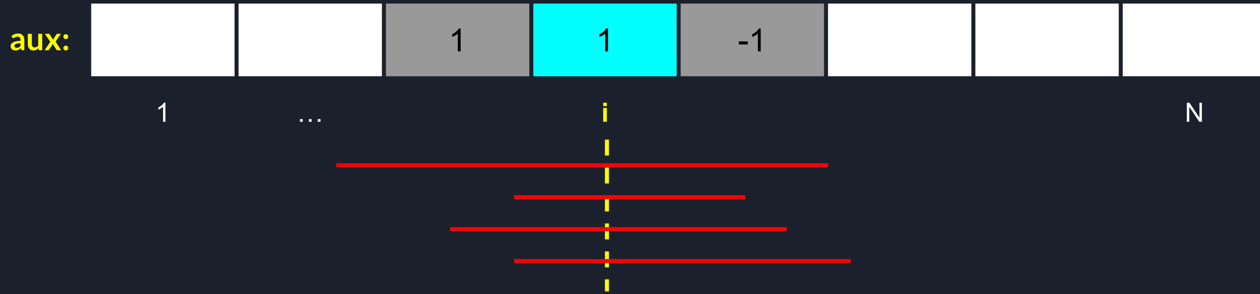
Ideia



- A dificuldade desse problema está em como verificar rápido se podemos trocar de 1 para -1
- O que temos que fazer é para cada restrição que contenha o índice i, teríamos que verificar se é possível trocar de 1 para -1. Ou seja, se mantivermos o quanto cada restrição excede do seu valor desejado, bastava passar por cada um e subtrair 2, se todos continuassem válidos (ou seja maiores ou iguais a zero) então podemos trocar de 1 para -1, senão não podemos (e desfazemos a operação de subtrair 2 em cada um deles)
- Mas obviamente isso daria TLE, como podemos fazer isso de forma eficiente ?

H - Lexical Sign Sequence

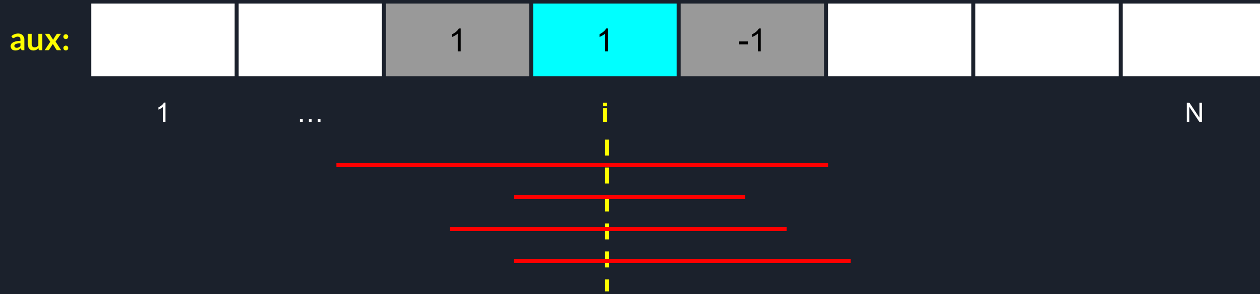
Ideia



- Podemos manter todos esses valores (pra cada restrição quanto ela excede do desejado) em um Set (na verdade multiset pois podemos ter valores iguais). Apenas para as restrições que passam por i.
- Para verificar se podemos trocar de 1 para -1, basta ver se o menor valor é maior ou igual a 2. Se não for, então deixamos 1 e seguimos para o próximo índice, mas se for então trocamos para -1 e devemos subtrair 2 de todos os valores presentes atualmente no Set. Mas como fazer isso ?

H - Lexical Sign Sequence

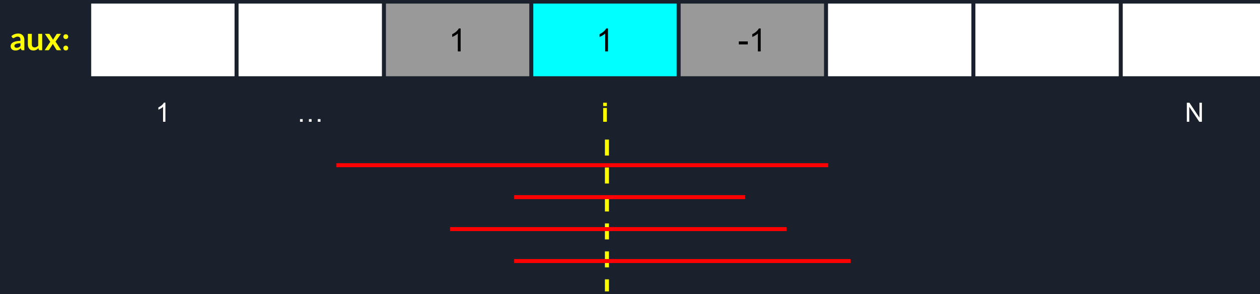
Ideia



- Para verificar se podemos trocar de 1 para -1, basta ver se o menor valor é maior ou igual a 2. Se não for, então deixamos 1 e seguimos para o próximo índice, mas se for então trocamos para -1 e devemos subtrair 2 de todos os valores presentes atualmente no Set. Mas como fazer isso ?
- Basta manter um valor auxiliar dizendo quanto devemos somar em cada elemento do Set. Assim para subtrair 2 de todo mundo basta subtrair 2 deste auxiliar.
- E quando for adicionar um determinado valor no Set, adicione o valor desejado MENOS o valor atual desta variável auxiliar.
- Costumo chamar esse truque de Set de Soma, pois é um Set com a operação de somar um determinado valor em todos os elementos.

H - Lexical Sign Sequence

Ideia

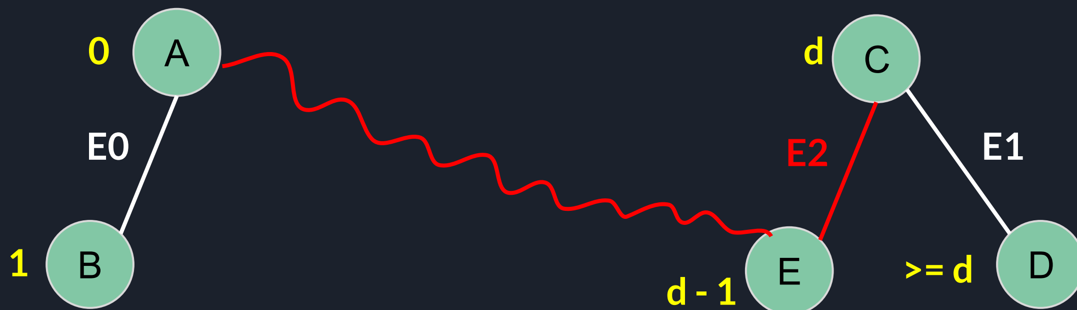


- No fundo estamos fazendo uma Line Sweep, onde nossa Line é um Set de Soma, e os eventos são entrada e saída de restrições. Mas fica mais simples de implementar mantendo pra cada índice uma lista dos índices das restrições que começam e terminam ali.
- Lembre-se de antes de processar o índice i, adicionar no Set os valores de todas as restrições que começam no índice i.
- E também remover do Set os valores de todas as restrições que terminam em i - 1
  - Mantenha salvo qual foi o valor adicionado realmente, ou seja, o valor que excede menos o valor da variável auxiliar no momento da entrada da restrição. Assim basta removê-lo do multiset (cuidado, remover usando find)

K - Boomerangs

Ideia

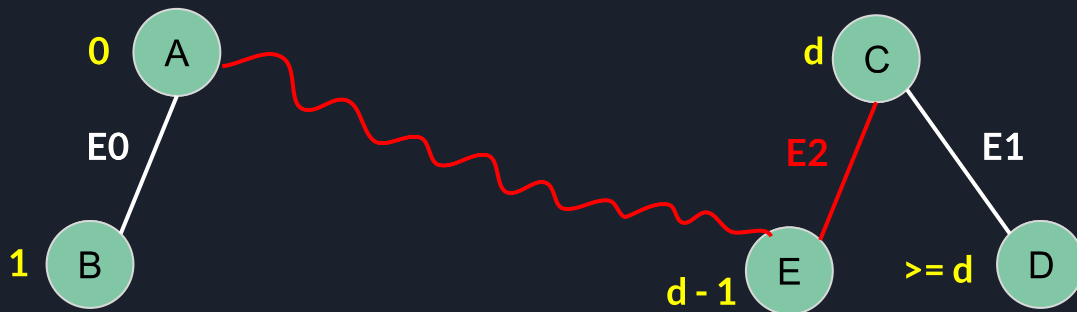
- Observação - Numa componente conexa com M arestas:
  - Se M for par haverá uma forma de colocar boomerangs que irá usar TODAS as arestas
  - Se M for ímpar, haverá uma forma que usará TODAS MENOS UMA aresta
- Para demonstrar que essa observação é verdadeira, imagine uma aresta não usada E0, e vamos supor que na mesma componente conexa há pelo menos mais uma outra aresta não usada. Imagine uma BFS a partir de algum dos vértices de E0, e seja E1 a aresta não usada que esteja mais próxima de E0 (menor distância de algum dos vértices).



K - Boomerangs

Ideia

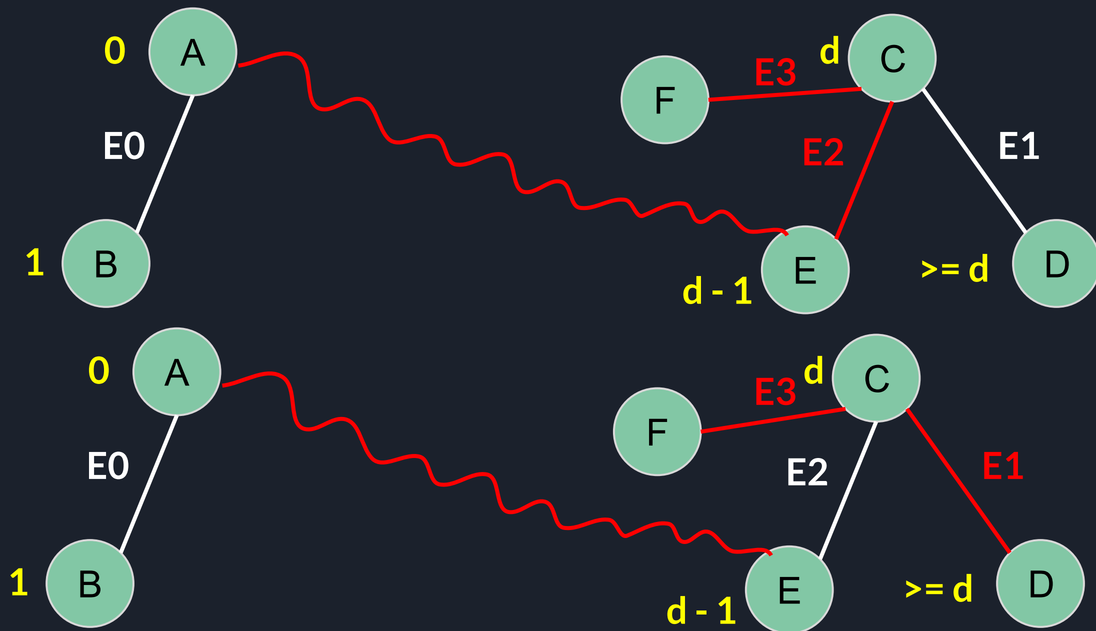
- Seja C o vértice de E1 com menor distância na BFS, digamos distância d
- Seja E o vizinho de C com distância d - 1 (quem o descobriu na BFS)
- Perceba que a aresta EC (na figura está como E2) tem que estar usada pois E1 é a não usada mais próxima de E0
- Vamos analisar dois casos, de acordo com qual aresta está formando um boomerang com E2
  - Caso 1: uma aresta de C
  - Caso 2: uma aresta de E



K - Boomerangs

Ideia

- Caso 1: uma aresta de C formando boomerang com E2. Podemos trocar E2 por E1

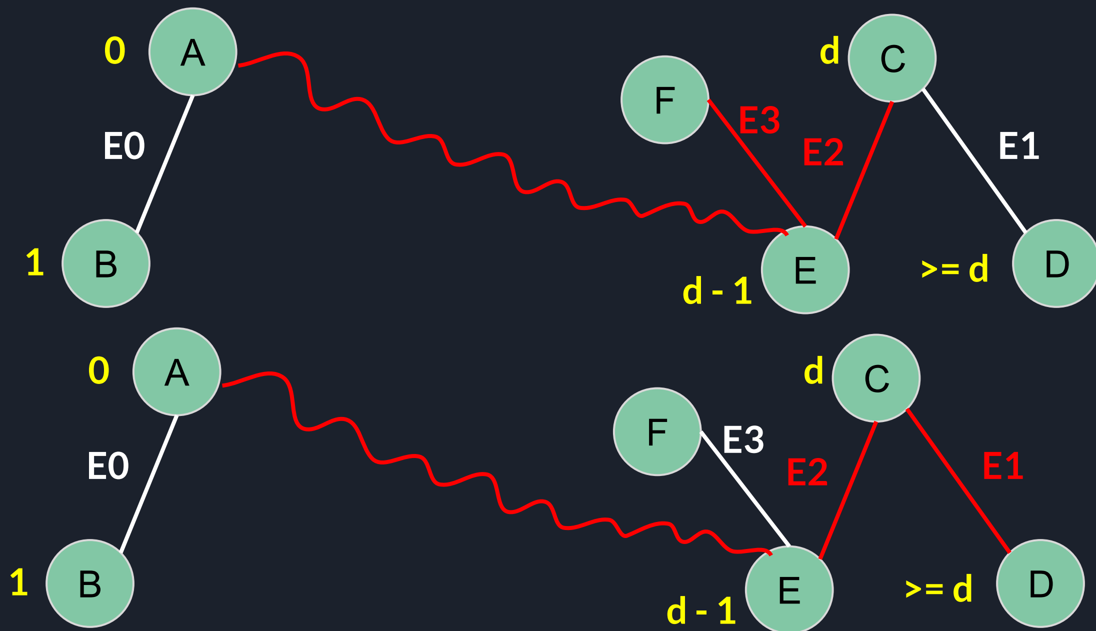


Agora a aresta não usada, E2, está mais próxima de E0

K - Boomerangs

Ideia

- Caso 2: uma aresta de E formando boomerang com E2. Podemos trocar E3 por E1



Agora a aresta não usada, E3, está mais próxima de E0

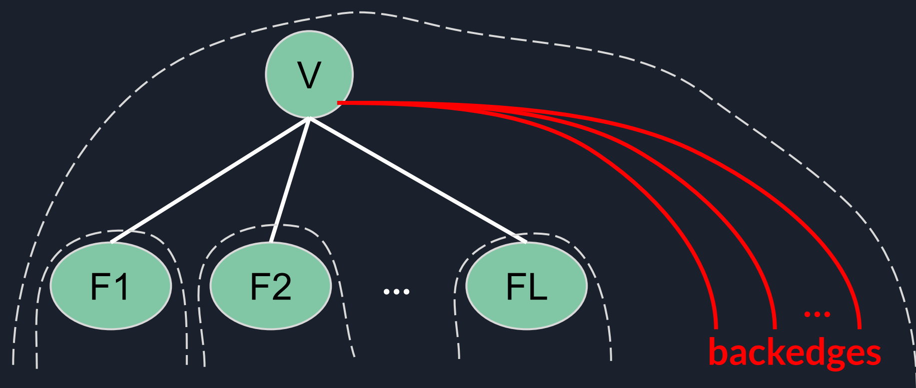
Ideia

- Desta forma mostramos que sempre que houver duas arestas não usadas numa mesma componentes, é possível trocas e deixá-las mais próximas. Se repetirmos esse processo, chegaremos num estado onde essas duas arestas não usadas possuem um vértice em comum, e portanto podemos formar um boomerang com elas
- Note que com isso, conseguimos demonstrar a observação inicial, que se a quantidade de arestas de uma componente for par, conseguiremos usar TODAS as arestas; e se for ímpar, conseguiremos usar TODAS MENOS UMA
- Esse algoritmo é BEEEM ineficiente, ele serve apenas para demonstrar a observação. Também poderia ser demonstrado diretamente pelo algoritmo final que discutiremos, mas é legal entender essa demonstração.
- Iremos chamar uma DFS para cada componente conexa do grafo, e essa DFS já irá construir a resposta, usando o seguinte algoritmo:

K - Boomerangs

Ideia

- Seja sube[V] a quantidade de arestas na subárvore de V (considerando a árvore da DFS). Considere uma aresta como parte da subárvore de V apenas se os dois vértices dessa aresta se encontram na subárvore de V.
- Dessa forma, as backedges pertencem ao vértice de menor profundidade na árvore da DFS (o de cima). Seja também back[V] a quantidade de backedges de V.
- Caso você nunca tenha visto o conceito de backedge, segue um tutorial bacana



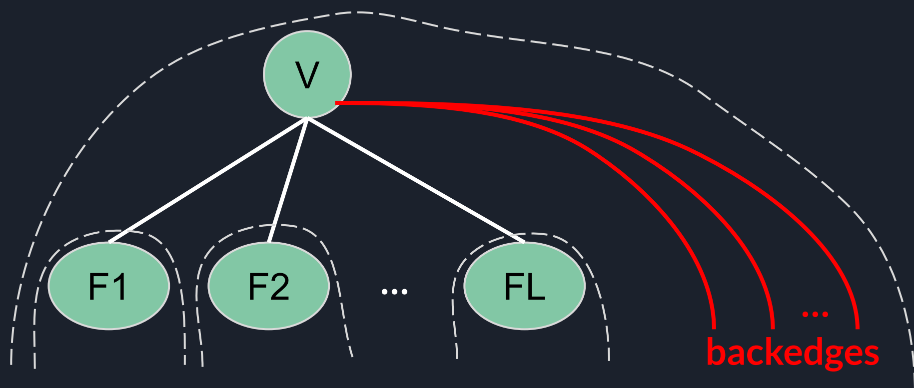
K - Boomerangs

Ideia

- Assim podemos escrever que:

$$sube[V] = \sum(1 + sube[Fi]) + back[V]$$

Portanto: $sube[V] = \sum(sube[Fi]) + L + back[V]$

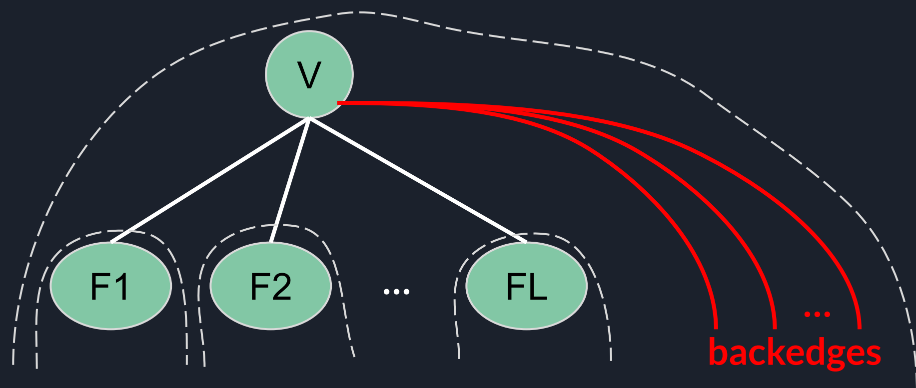


Dessa forma conseguimos calcular sube[V] durante a DFS.
Note que para calcular back[], basta toda vez que encontrar uma backedge, somar 1 no back do vértice mais acima

K - Boomerangs

Ideia

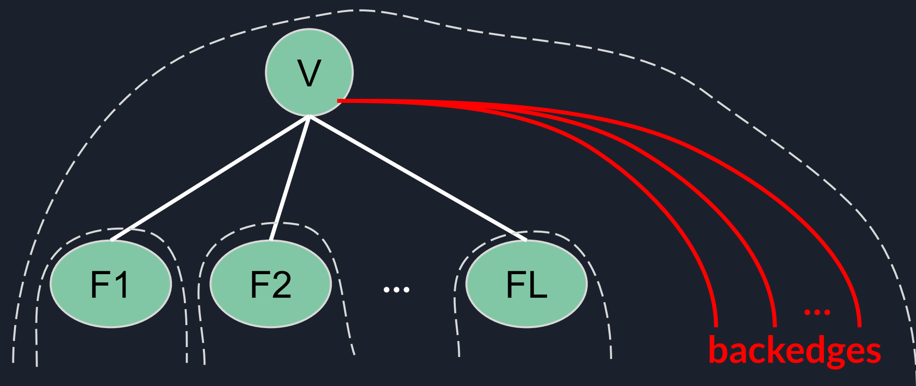
- Durante a DFS, mantenha para cada vértice uma LISTA DE ARESTAS. Essas serão as arestas onde ele será o MEIO do boomerang. Inicialmente essa lista está vazia para todos os vértices.
- Ao final da DFS, para construir a resposta, basta para cada vértice, parear como quiser as arestas em sua lista.
- Caso sube[V] seja par, TODAS as listas de arestas dos vértices na subárvore de V terão quantidade de arestas par, o que significa que TODAS serão usadas.



K - Boomerangs

Ideia

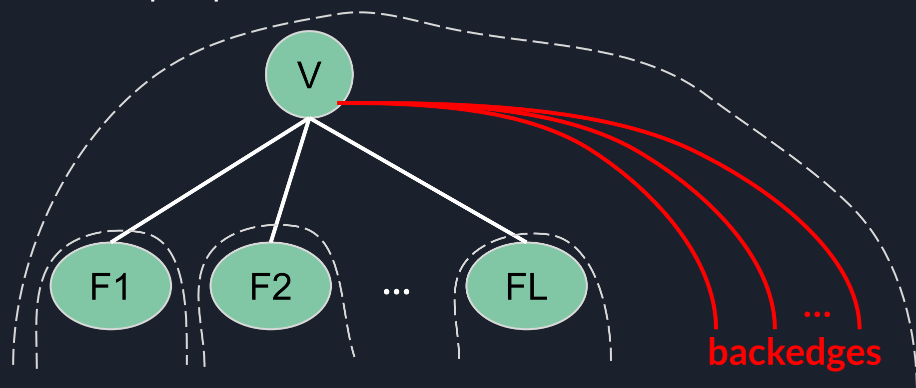
- Caso sube[V] seja ímpar, TODAS as listas de arestas dos vértices na subárvore de V terão quantidade de arestas par, E APENAS V terá quantidade ímpar. Ou seja, deixaremos APENAS uma aresta não usada, e ela será alguma aresta da raiz da subárvore, no caso V.



K - Boomerangs

Ideia

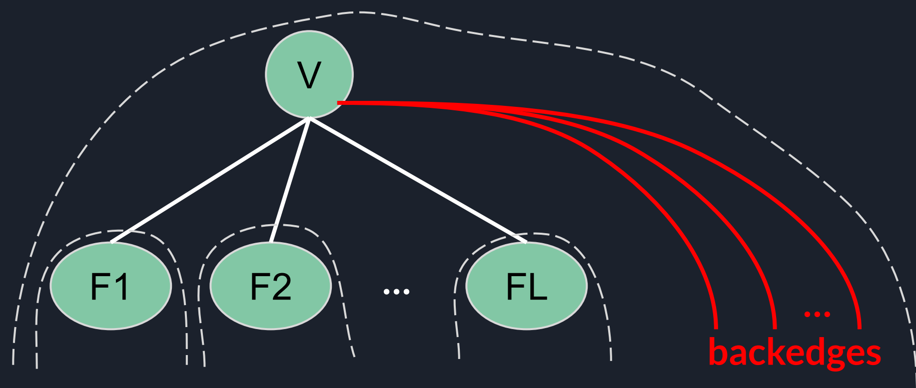
- Algoritmo:
  - Adicione TODAS as backedges de V à lista de aresta de V (back[V])
  - Se sube[Fi] for PAR, então adicione a aresta (V, Fi) à lista de V (de Fi pra baixo, todas as listas já estarão PAR)
  - Se sube[Fi] for ÍMPAR, então adicione a aresta (V, Fi) à lista de Fi (apenas a lista de Fi estava ÍMPAR, dessa forma estamos deixando todas as listas de Fi pra baixo como PARES)
- Pela forma como estamos montando essas listas de arestas, irá funcionar, vamos ver um exemplo do porque funciona:



K - Boomerangs

Ideia

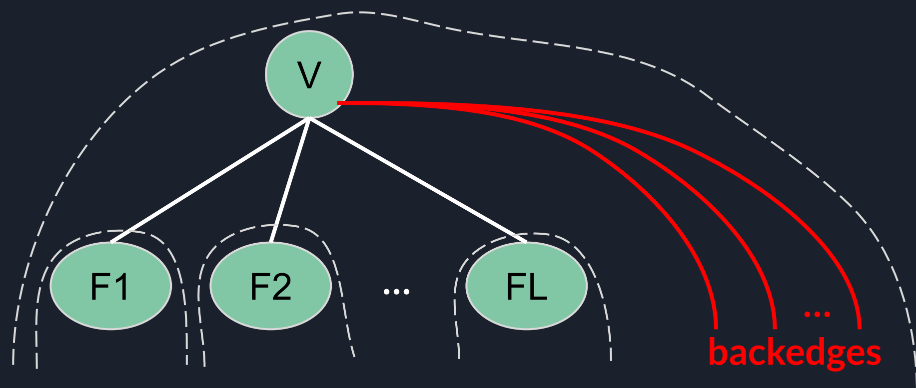
- Temos: sube[V] = ∑(sube[Fi]) + L + back[V]
- Vamos analisar o caso onde sube[V] é par, L é par e back[V] é par.
    - Nesse caso, ∑(sube[Fi]) é par, portanto A QUANTIDADE de filhos onde sube[Fi] é ímpar, é PAR (em outras palavras, pra que a soma dê PAR, a quantidade de termos ÍMPARES é PAR)
    - Portanto sobrará uma quantidade PAR de arestas na lista de V: L - #FilhosImpares + back[V]



K - Boomerangs

Ideia

- Temos: sube[V] = ∑(sube[Fi]) + L + back[V]
- Vamos analisar o caso onde sube[V] é par, L é ímpar e back[V] é par.
  - Nesse caso, ∑(sube[Fi]) é ímpar, portanto A QUANTIDADE de filhos onde sube[Fi] é ímpar, é ÍMPAR (em outras palavras, pra que a soma dê ÍMPAR, a quantidade de termos ÍMPARES é ÍMPAR)
  - Portanto sobrará uma quantidade PAR de arestas na lista de V: L - #FilhosImpares + back[V]
- Se analisar todos os casos, verá que sempre funciona



BOA SUMMER
SCHOOL!
Aproveitem