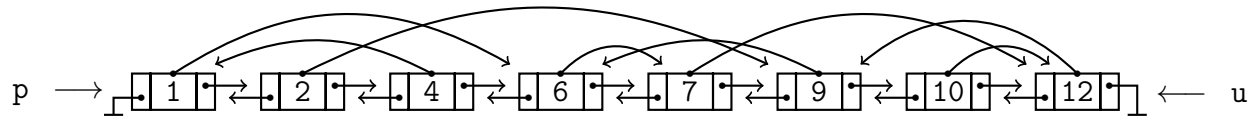
A lista com ponteiros aleatórios

Nós acabamos de ver que a lista duplamente encadeada nos dá mais flexibilidade para fazer as coisas.

Ora, mas se isso é o caso, então agora nós podemos considerar uma lista triplamente encadeada.

Quer dizer, a ideia é que nós vamos ter os ponteiros que apontam para frente e para trás.

E vamos ter um terceiro ponteiro que aponta para um lugar aleatório na lista



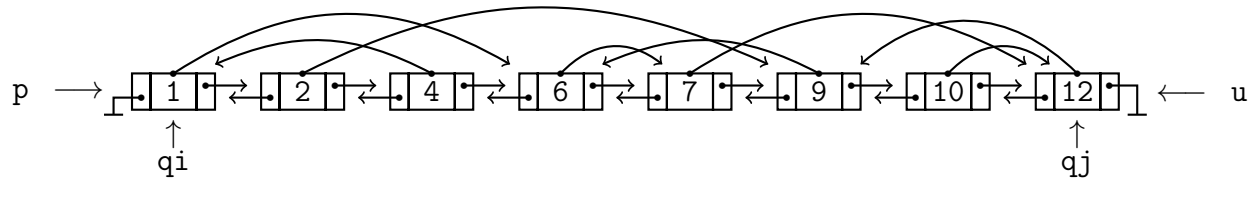
Mas, pra que isso?

Bom, a ideia é que os ponteiros para frente e para trás nos permitem andar pela lista dando passos pequenos.

E os ponteiros aleatórios permitem dar saltos grandes, para chegar mais rápido a um local distante.

Daí, a estratégia de busca é semelhante à busca binária.

Quer dizer, nós começamos colocando dois ponteiros q_i e q_j no primeiro e no último elementos da lista



A cada passo o ponteiro q_i tem 3 lugares para ir

- o próximo elemento
- o elemento apontado pelo ponteiro aleatório de q_i
- o elemento apontado pelo ponteiro aleatório de q_j

E nós escolhemos aquele que está mais próximo do número `k` — (*mas não excede esse valor*)

O ponteiro q_j se movimenta da mesma maneira — (*na direção contrária ...*)

A coisa fica assim

```
func busca-PA (k,p,u)

      Se ( p == Nulo )   Retorna (Nulo)                          // Não está lá ...

      qi = p;   qj = u                                           // posiciona os dedos
      Enquanto ( qi->num ≤ qj->num )
            Se ( qi->num == k )   Retorna (qi)
            Se ( qj->num == k )   Retorna (qj)
            qi = Maior-menor (k,qi->prox,qi->ale,qj->ale)        // avança qi
            qj = menor-Maior (k,qi->prox,qi->ale,qj->ale)        // avança qj

      Retorna (p)
```

onde

- a função `Maior-menor()` retorna o ponteiro que aponta para o maior valor menor ou igual a `k` — (*ou então* `prox`)
- a função `menor-Maior()` retorna o ponteiro que aponta para o menor valor maior ou igual a `k` — (*ou então* `ant`)