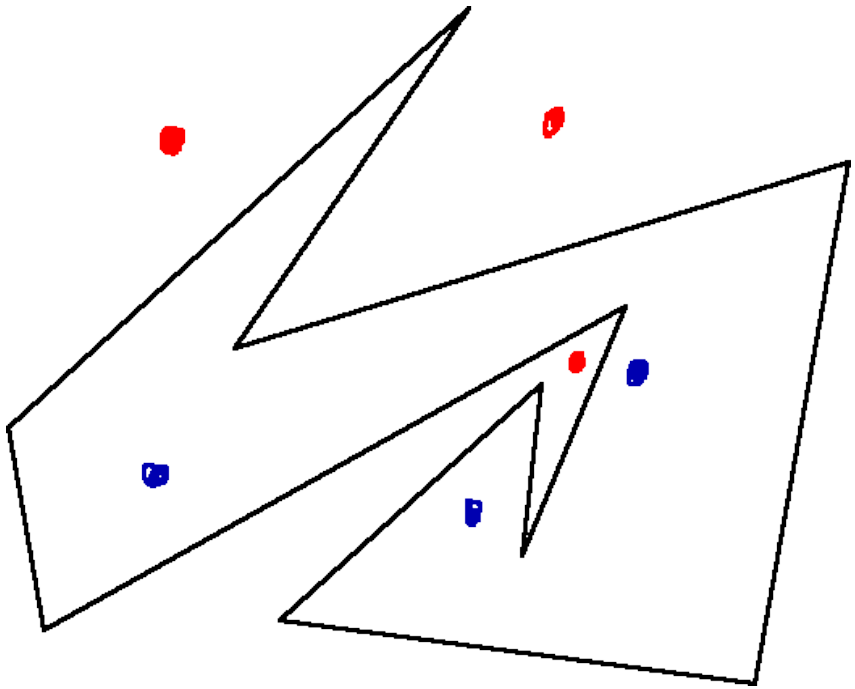
Perzistentní datová struktura:

- dynamická
- přístup k předchozím verzím

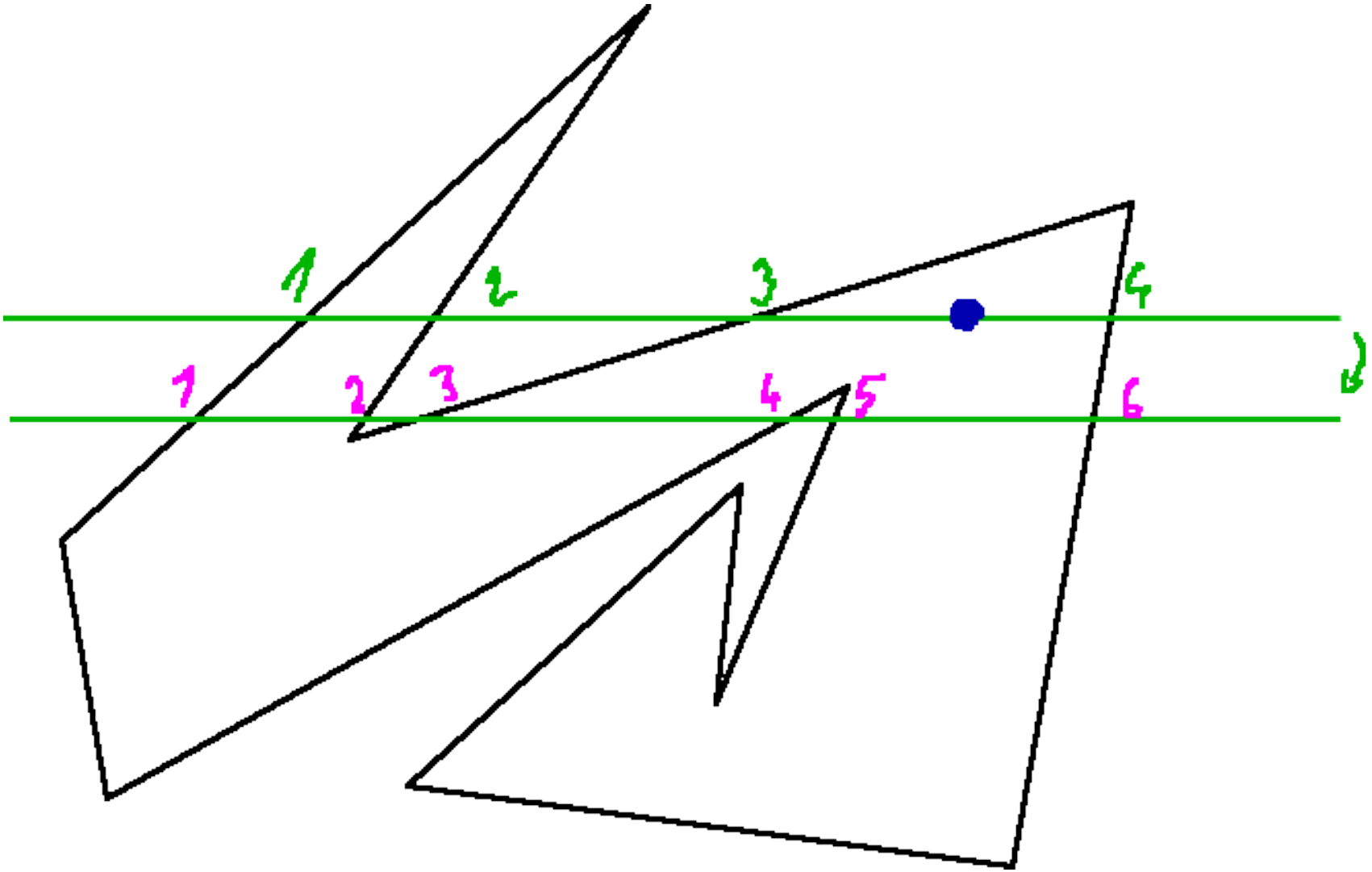
Zadání

Máme (ne nutně konvexní) mnohoúhelník v rovině. Pro zadané body $b_1, \ldots, b_n$ rozhodněte, zda leží uvnitř něj.



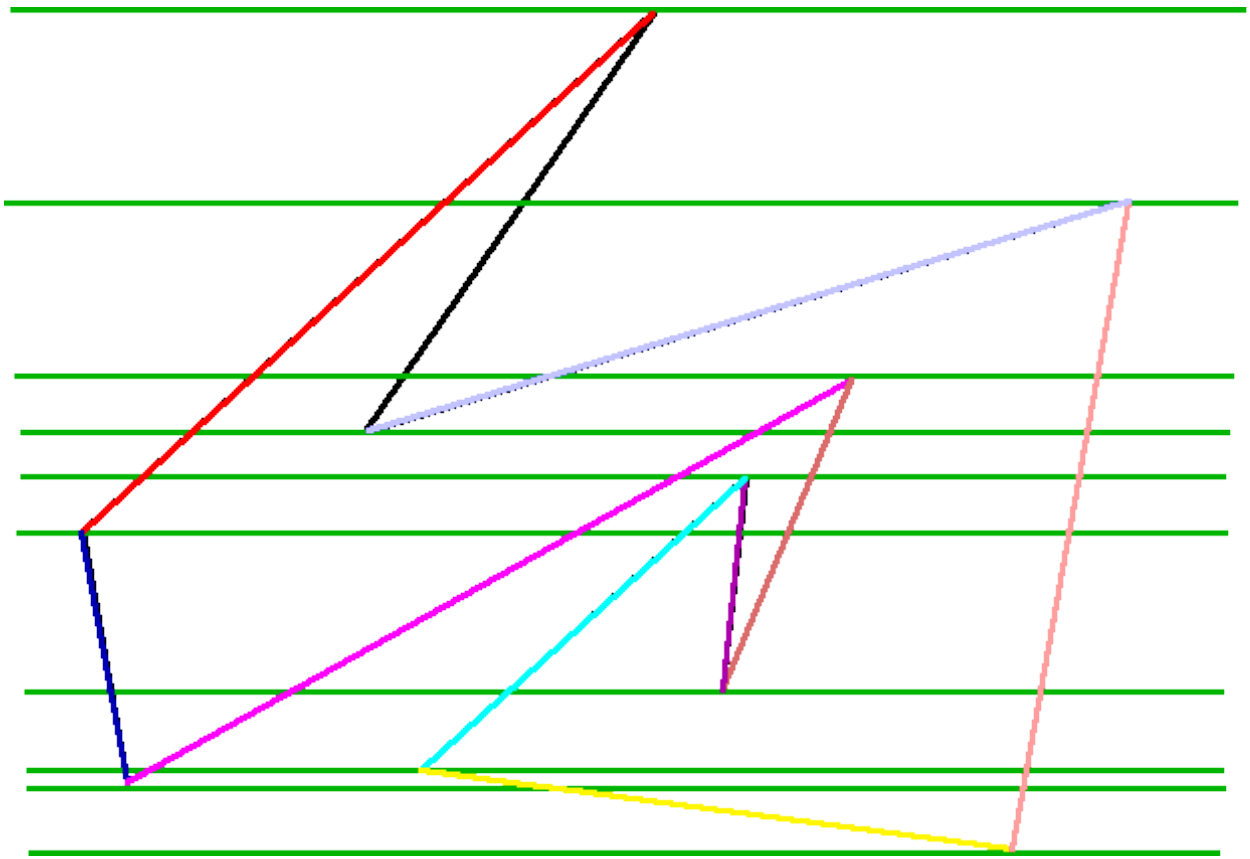
Řešení když známe $b_1, \ldots, b_n$ předem:

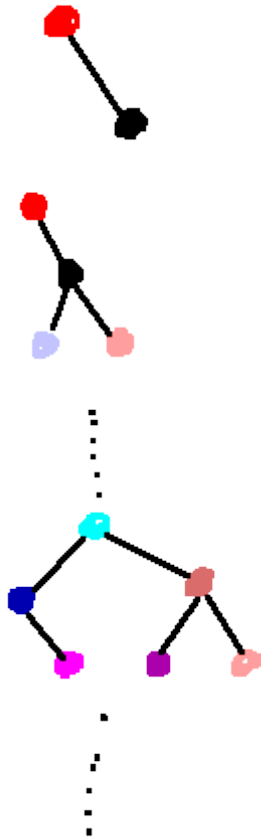
- Udržujeme ve stromu úsečky přes scanline zleva doprava.
- Dotaz: Počet úseček nalevo od b_i (je sudý nebo lichý)?



$b_1, \ldots, b_n$ dostáváme postupně a musíme okamžitě odpovídat:

- Pamatujeme si všechny verze stromu, dohledáme relevantní půlením intervalů.
- Nechceme stromy kopírovat celé!
- Řešení: Perzistentní strom.





Základní idea: Nic nepřepisovat, vytvářet nové verze.

Vložení do stromu:

```
insert (T, x):
  if null(T): return new tree(x);
  if (x > T->value)
    T->right = insert (T->right, x);
  else
    T->left = insert (T->left, x);
  return T;
```

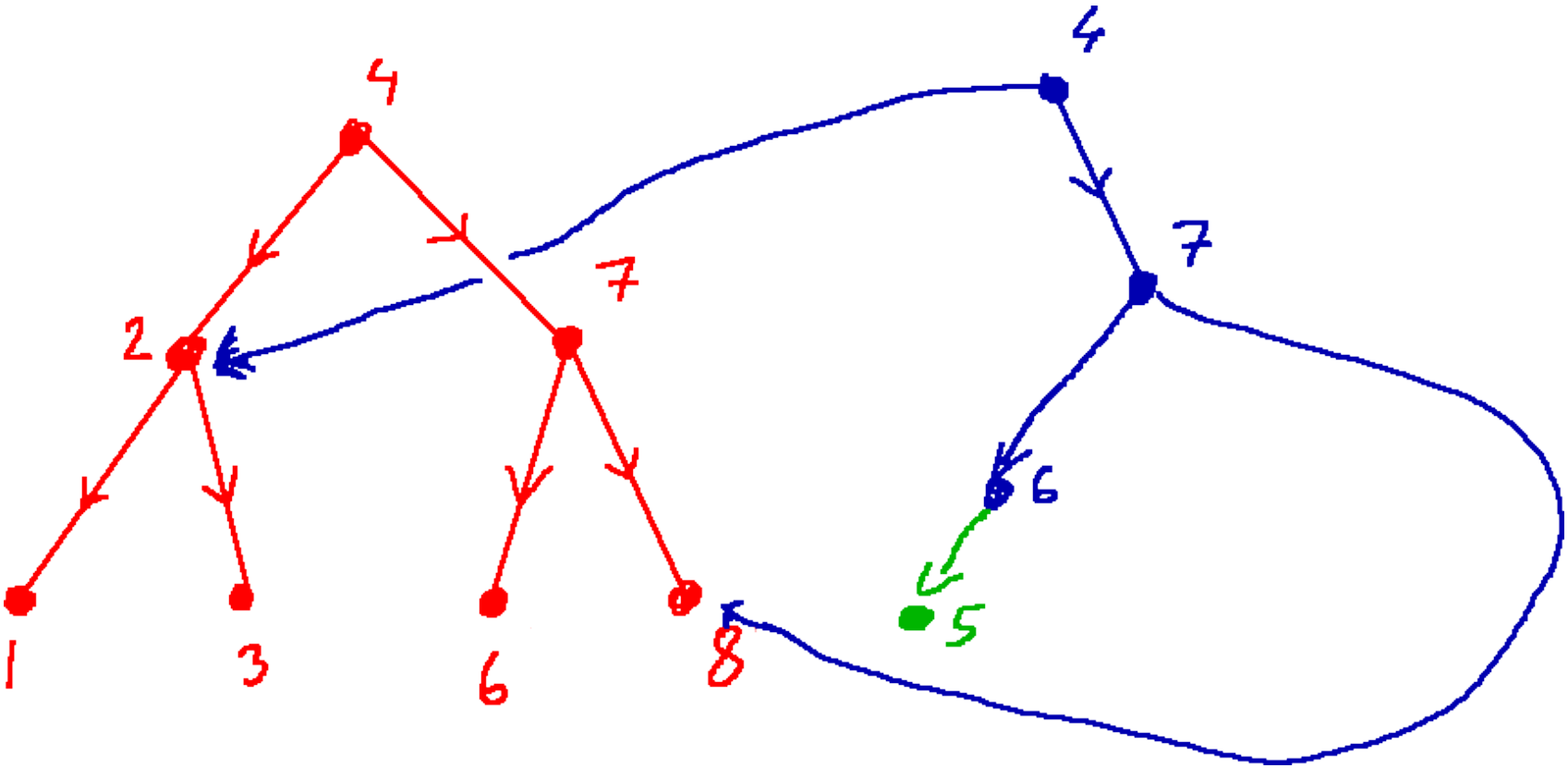
Základní idea: Nic nepřepisovat, vytvářet nové verze.

Vložení do stromu:

```
insert (T, x):
  if null(T): return new tree(x);
  ret = new tree (T->value);
  if (x > T->value)
    ret->left = T->left;
    ret->right = insert (T->right, x);
  else
    ret->left = insert (T->left, x);
    ret->right = T->right;
  return ret;
```

Základní idea: Nic nepřepisovat, vytvářet nové verze.

Vložení do stromu:



Obdobně:

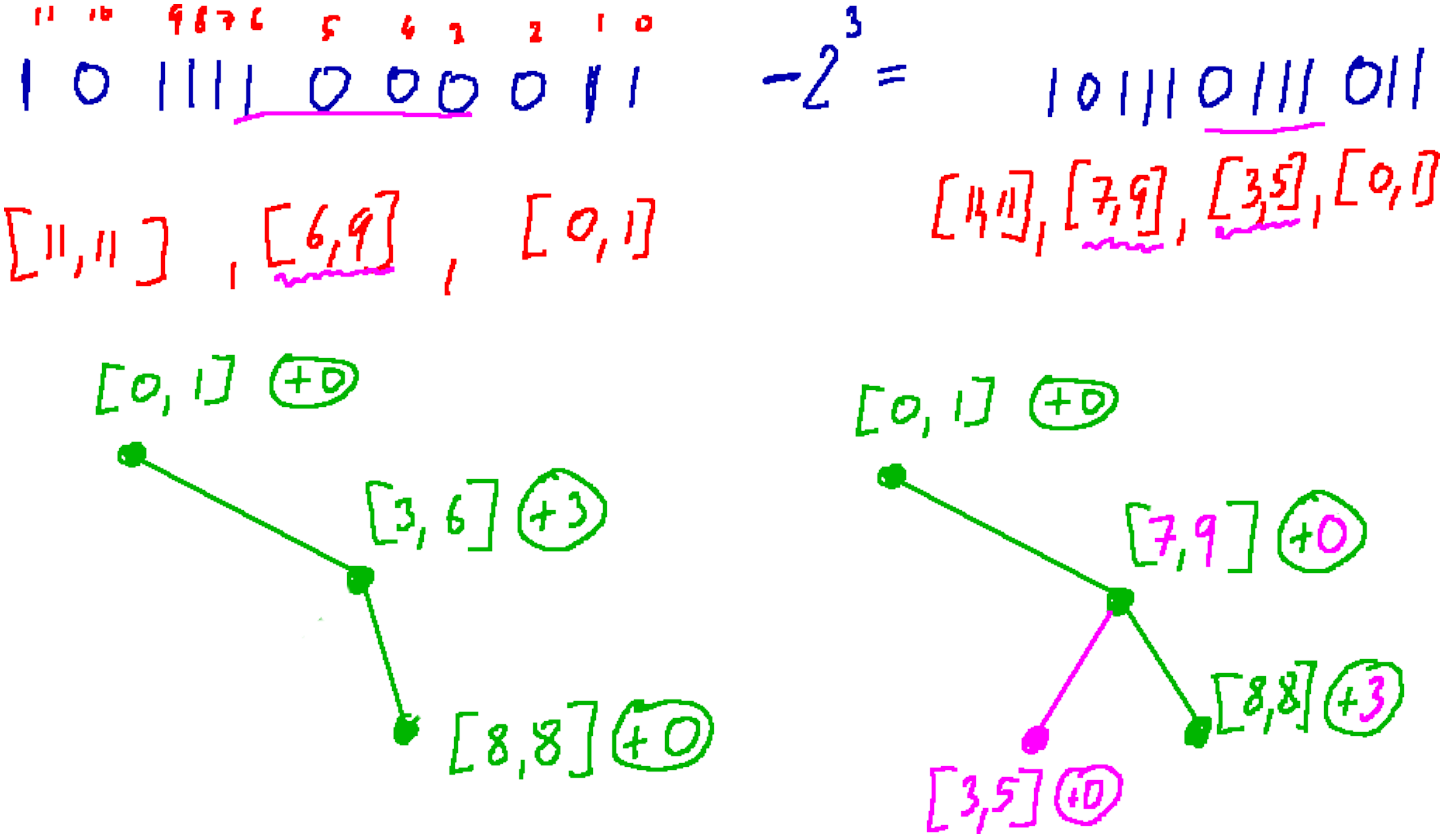
- Smazání ze stromu.
- Vyvažování.

Zadání

Na dlouhých číslech implementujte efektivně násobení 2, odečtení 2^k a kopírování.

Reprezentace:

- Strom intervalů číslic 1 v zápisu.
- Líná informace: Vše v podstromu posunout o k.



Nefunguje, pokud k měněnému místu vede dlouhá cesta.
Obecné řešení:

- Nahradit ukazatele indexy v poli a
- pole nahradit perzistentním stromem.

Zhorší složitost log(počet zápisů)-krát.

Příklad: seznam, zápis na konec:

```
konec->next = new element (x, NULL);
konec = konec->next;
```

Nefunguje, pokud k měněnému místu vede dlouhá cesta.
Obecné řešení:

- Nahradit ukazatele indexy v poli a
- pole nahradit perzistentním stromem.

Zhorší složitost log(počet zápisů)-krát.

Příklad: seznam, zápis na konec:

```
new_id = last_id++;
elts[konec_id].next_id = new_id;
elts[new_id].value = x;
elts[new_id].next_id = -1;
konec_id = new_id;
```

Nefunguje, pokud k měněnému místu vede dlouhá cesta.
Obecné řešení:

- Nahradit ukazatele indexy v poli a
- pole nahradit perzistentním stromem.

Zhorší složitost log(počet zápisů)-krát.

Příklad: seznam, zápis na konec:

```
new_id = Q.last_id;
NEW_Q.last_id = new_id + 1;
konec = find (Q.elts, Q.konec_id);
NEW_Q.elts = replace (Q.elts, Q.konec_id,
                      new element (konec->value, new_id));
NEW_Q.elts = insert (NEW_Q.elts, new_id,
                      new element (x, -1));
NEW_Q.konec_id = new_id;
```

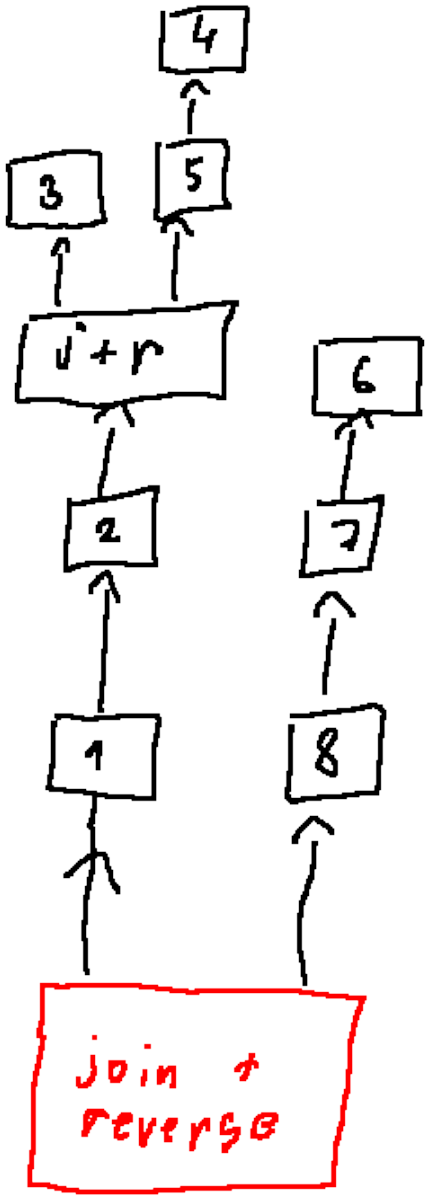
Nefunguje pro amortizované struktury.

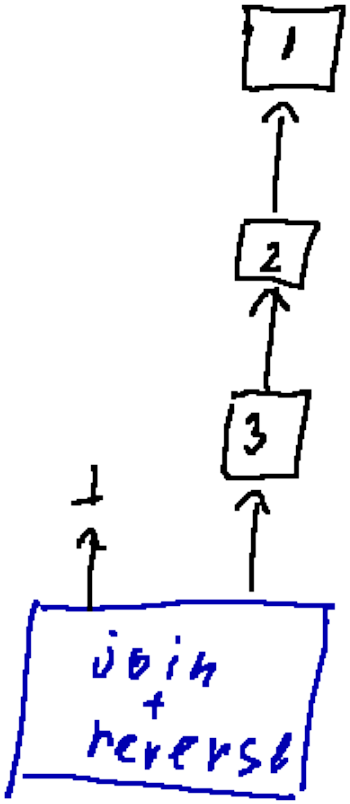
- Možnost vynutit opakování pomalé operace.
- Řešení:
  - Risknout to.
  - Vyhnout se amortizaci.
  - Složitější amortizace (s propagací do starších verzí).

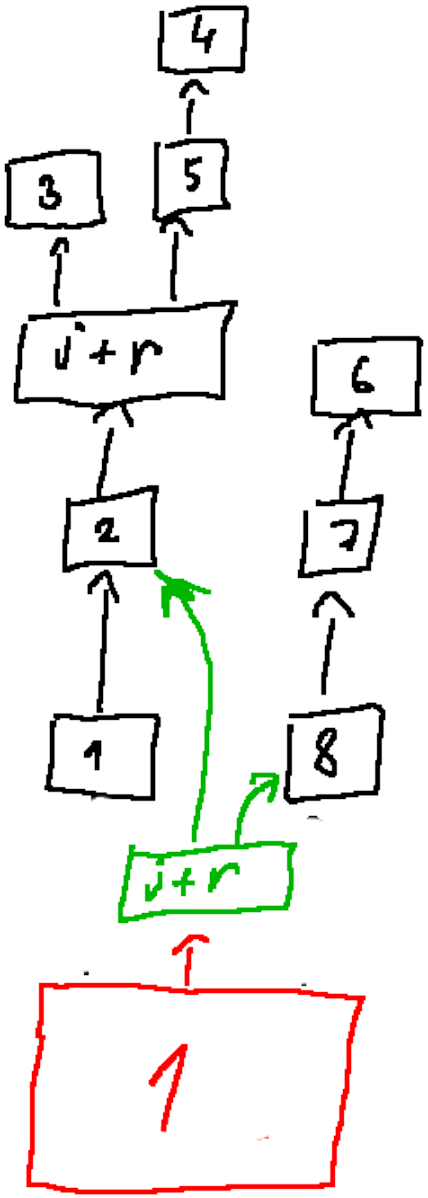
Příklad: Fronta ze dvou zásobníků.

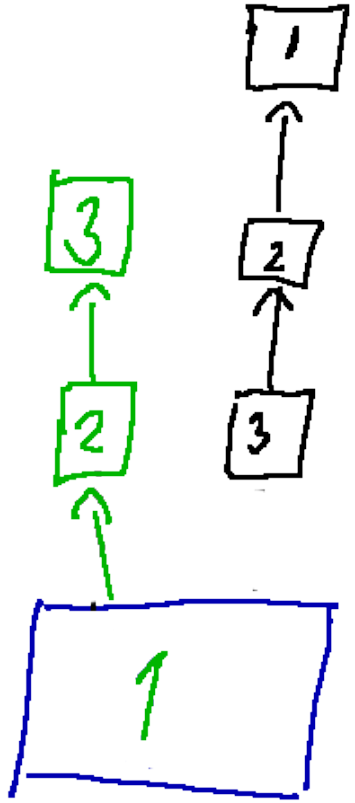
```
enter (x):
  levy = new element (x, levy);

leave:
  if null(pravy):
    pravy = reverse (levy);
    levy = NULL;
  ret = pravy->value;
  pravy = pravy->next;
  return ret;
```

```
struct seznam {
  bool evaluated;
  int val; seznam *a, *rev;

  get (int &v, seznam *&rest):
      if (!evaluated):
         if (!a):
            reverse (rev)->get (val, a);
         else:
            a->get (val, tmp);
            a = new seznam (tmp, rev);
         evaluated = true;
      v = val; rest = a;
};
```

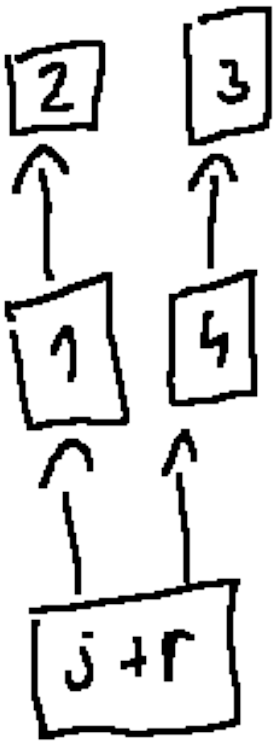

4
3
5
j+r
6
2
7
1
8
join + reverse

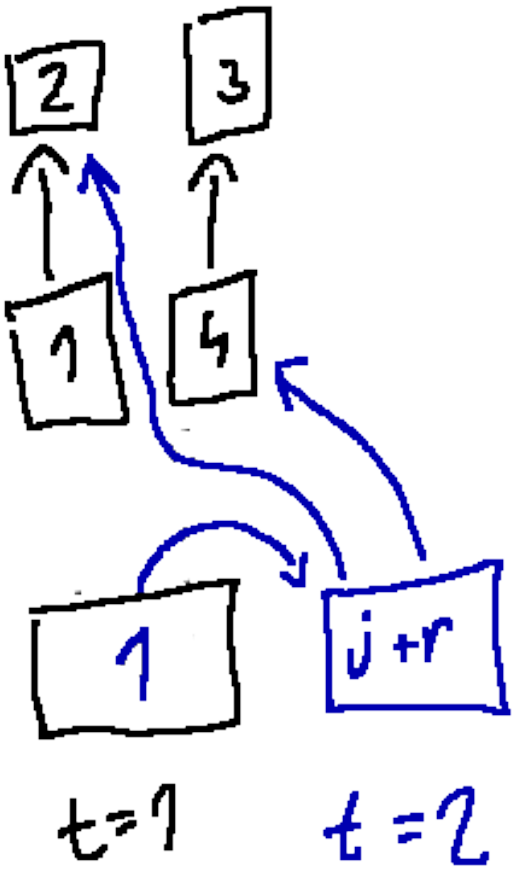

1
2
3
join + reverse

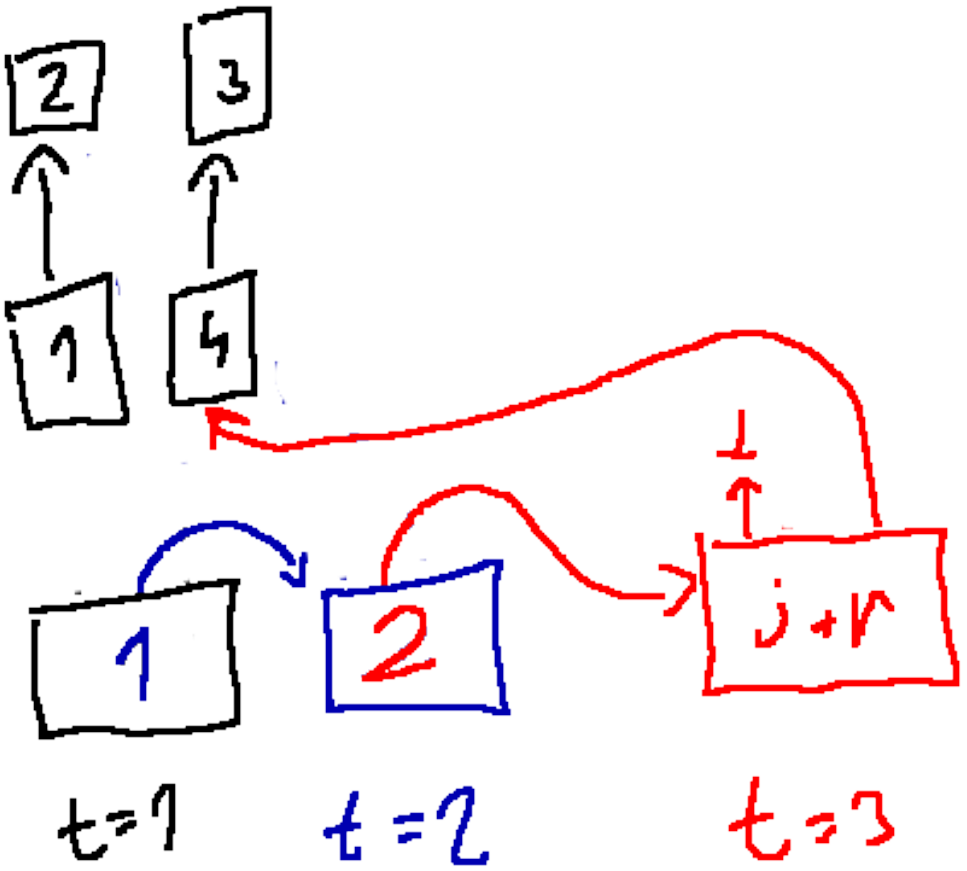

4
3
5
j+r
6
2
7
1
8
j+r
1

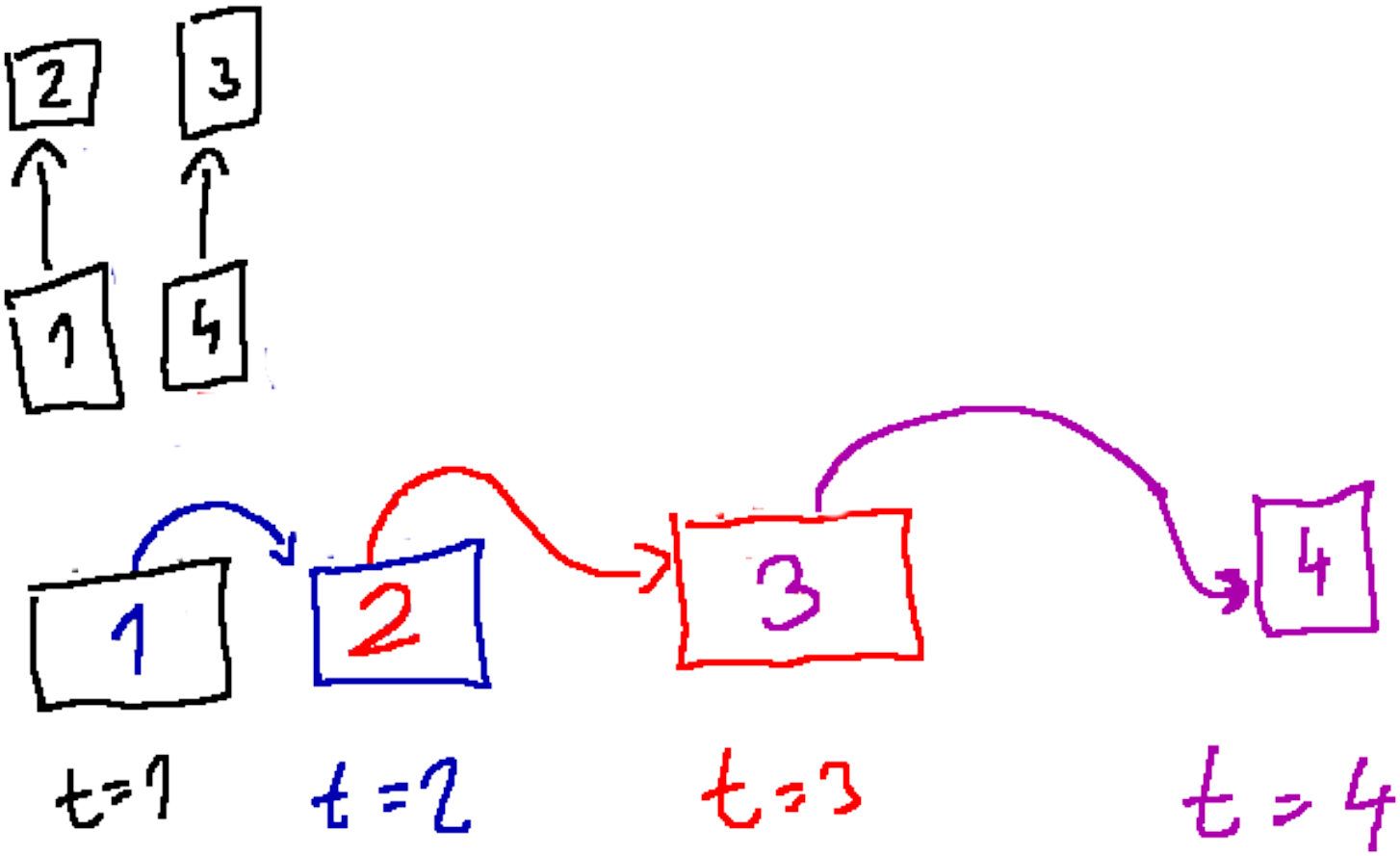

1
3
2
2
3
1

```
struct queue {
  seznam *levy, *pravy;
  int nlevy, npravy;
  balance (void):
     if (nlevy <= npravy) return;
     npravy += nlevy; nlevy = 0;
     pravy = new seznam (pravy, levy); levy = NULL;
};
enter (queue *q, int x):
  queue *nw = new queue (*q);
  nw->nlevy++; nw->levy = new seznam (x, q->levy);
  nw->balance ();
  return nw;
leave (queue *q, int &ret):
  queue *nw = new queue (*q);
  nw->pravy--; nw->pravy->get (ret, nw->pravy);
  nw->balance ();
  return nw;
```


2
3
1
4
j+r
t=1


2
3
1
4
1
j+r
t=1
t=2


2
3
1
4
1
2
j+1
1
t=1
t=2
t=3


2
3
1
4
1
2
3
4
t=1
t=2
t=3
t=4