Hlaupabil

Bergur Snorrason

4. apríl 2022

Hlaupabil

- Aðferð *hlaupabila* (e. *sliding window*) er stundum hægt að nota til að taka dæmi sem hafa augljósa $\mathcal{O}(n^2)$ lausn og gera þau $\mathcal{O}(n)$ eða $\mathcal{O}(n \log n)$.

- Skoðum dæmi:
- Gefið n, k og svo n tölur a_i, þ.a. $a_i \in \{0, 1\}$ finndu lengd lengstu bilanna í rununni $(a_n)_{n\in\mathbb{N}}$ sem innihelda bara 1 ef þú mátt breyta allt að k tölum.
- Sjáum strax að maður vill alltaf breyta 0 í 1 og aldrei öfugt.
- Sjáum því að við erum að leita að lengstu bilunum í $(a_n)_{n\in\mathbb{N}}$ sem hefur í mesta lagi k stök jöfn 0.
- Gefum okkur nú hlaupabil. Það byrjar tómt.
- Við löbbum svo í gegnum $(a_n)_{n\in\mathbb{N}}$ og lengjum bilið að aftan.
- Ef það eru einhvern tímann fleiri en k stök í bilinu sem eru 0 þá minnkum við bilið að framan þar til svo er ekki lengur.

```
k = 2
l = 0
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|
```

```
k = 2
l = 1
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
| |
```

```
k = 2
l = 2
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|   |
```

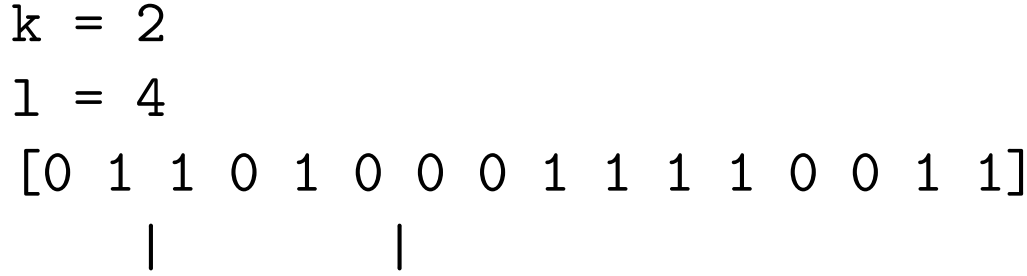
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|     |
```

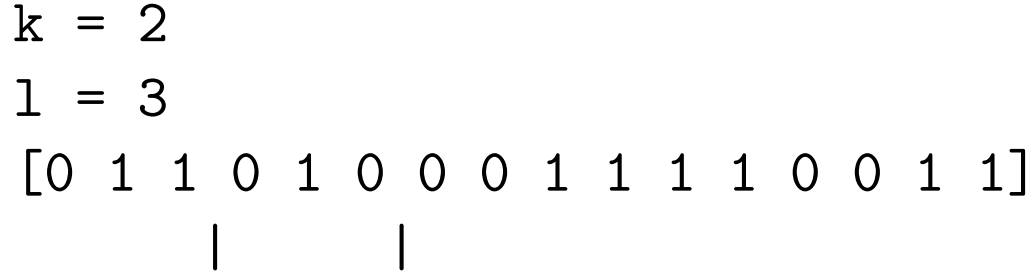
```
k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|       |
```

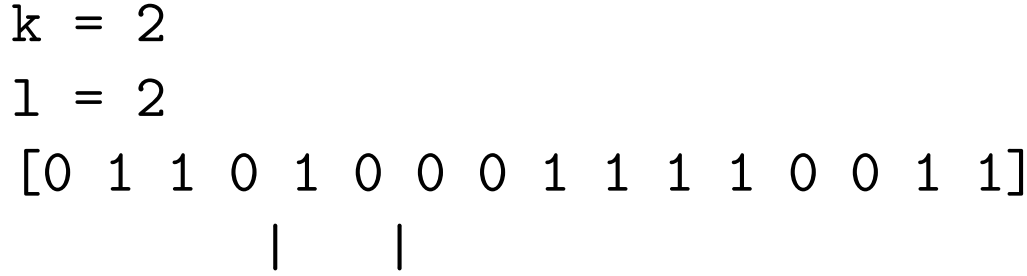
```
k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|         |
```

```
k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
  |       |
```

```
k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
  |           |
```


k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 2
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
| |

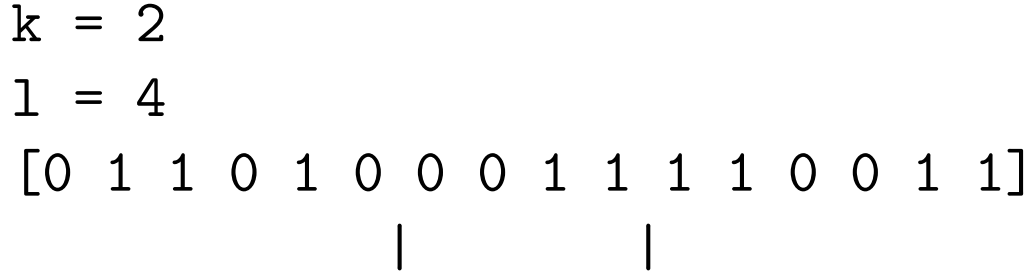
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
         |       |
```

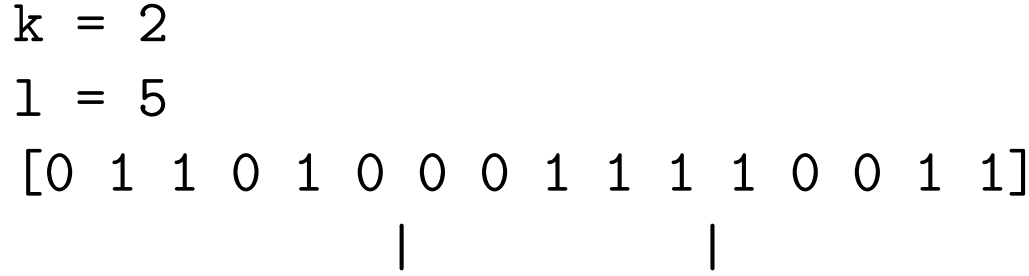
```
k = 2
l = 2
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
           |   |
```

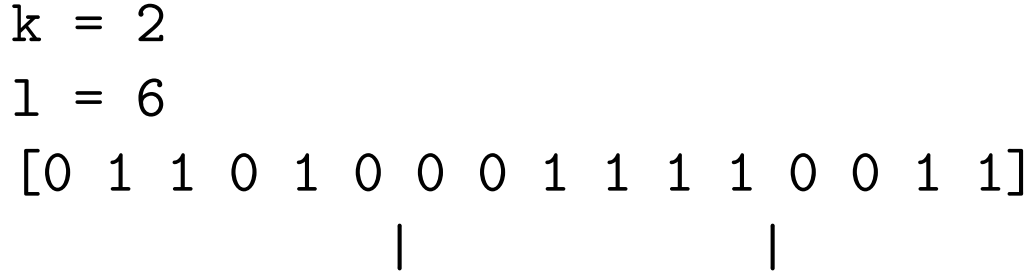
```
k = 2
l = 1
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
             | |
```

```
k = 2
l = 2
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
             |   |
```

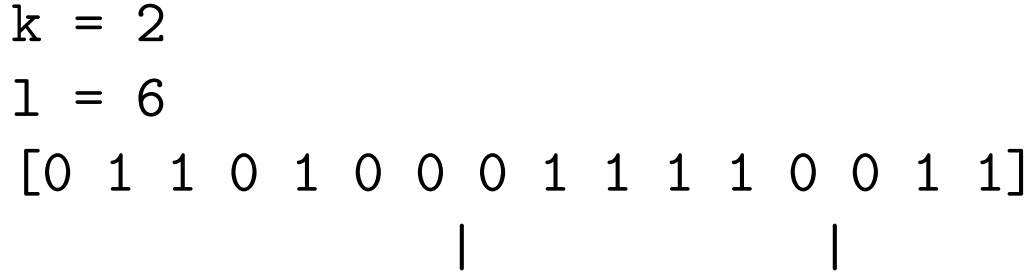
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
             |       |
```

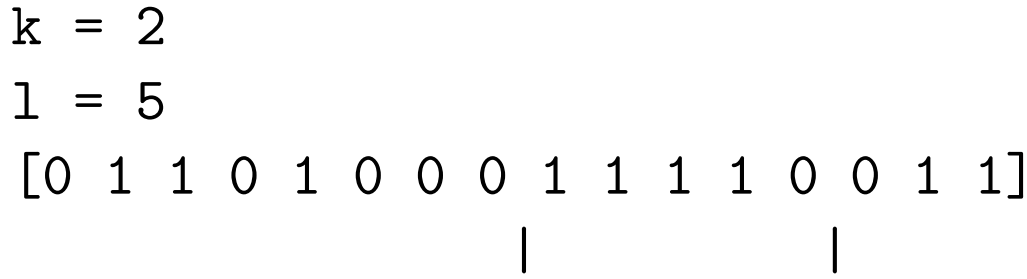

k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

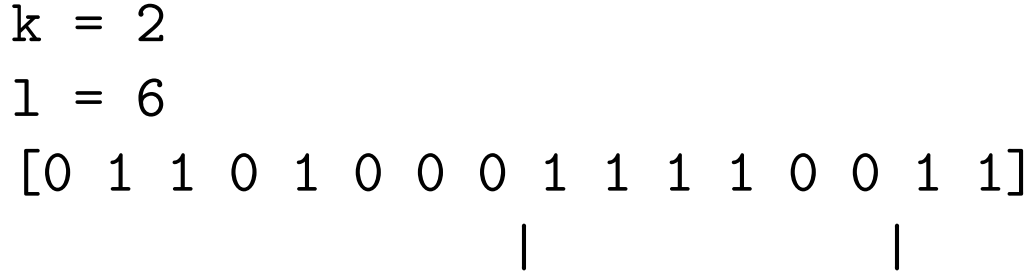

k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

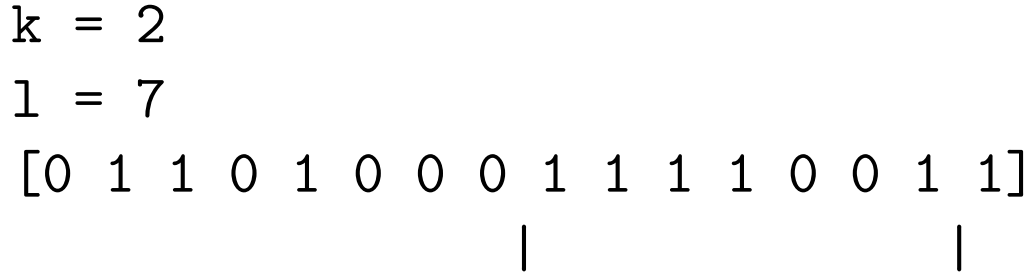

k = 2
l = 6
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

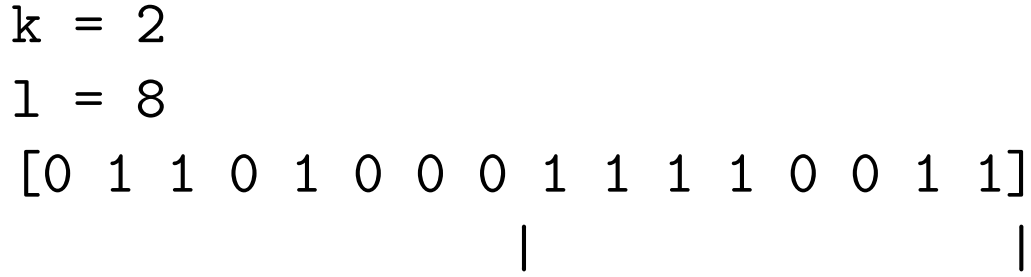
```
k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
                  |         |
```


k = 2
l = 6
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 6
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 7
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]


k = 2
l = 8
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

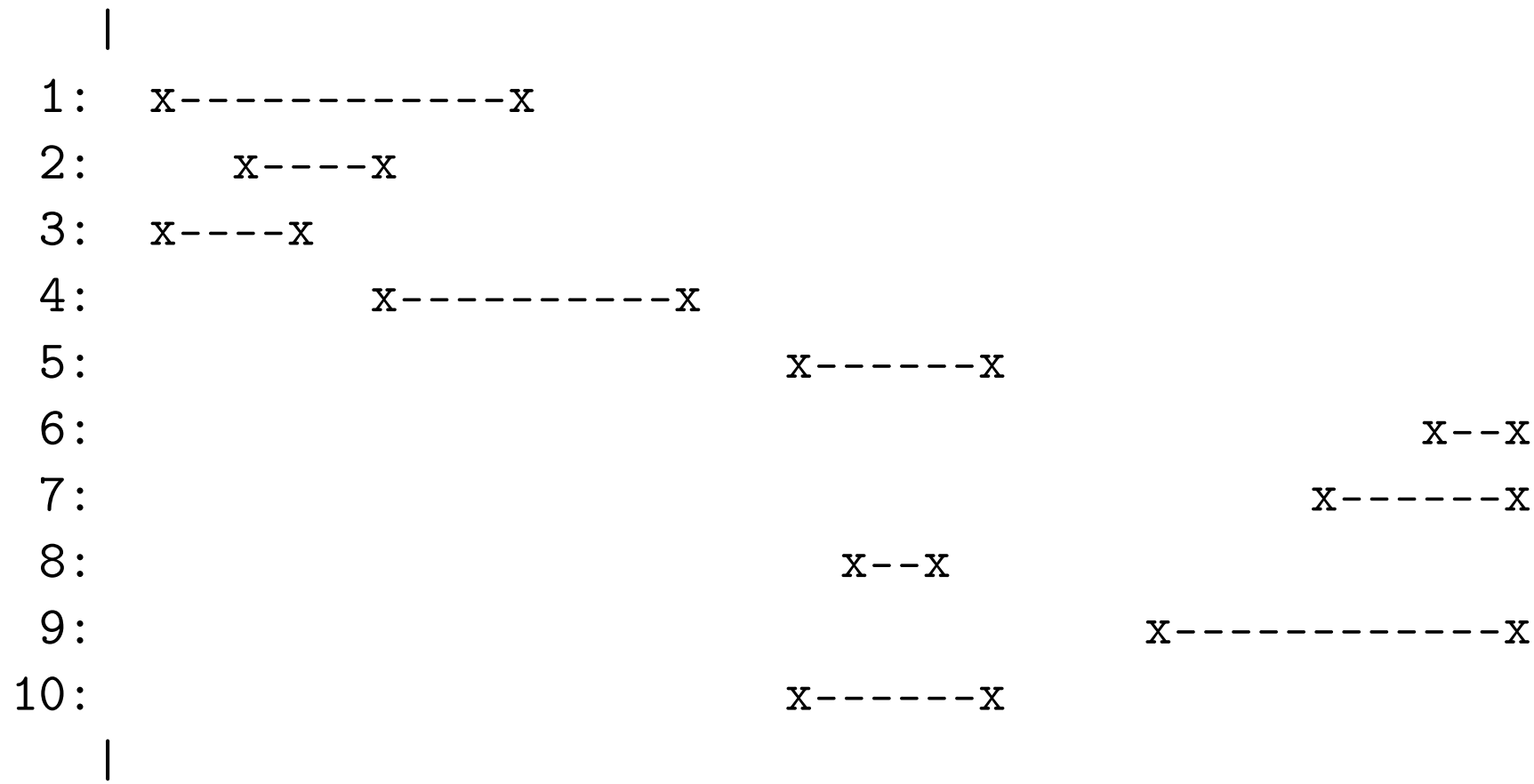
```c
int main()
{
    int n, k, i;
    scanf("%d%d", &n, &k);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &(a[i]));
    int b = 0, z = 0, mx = 0;
    for (i = 0; i < n; i++)
    {
        if (a[i] == 0) z++;
        while (z > k)
        {
            if (a[b] == 0) z--;
            b++;
        }
        if (i - b + 1 > mx) mx = i - b + 1;
    }
    printf("%d\n", mx);
    return 0;
}
```

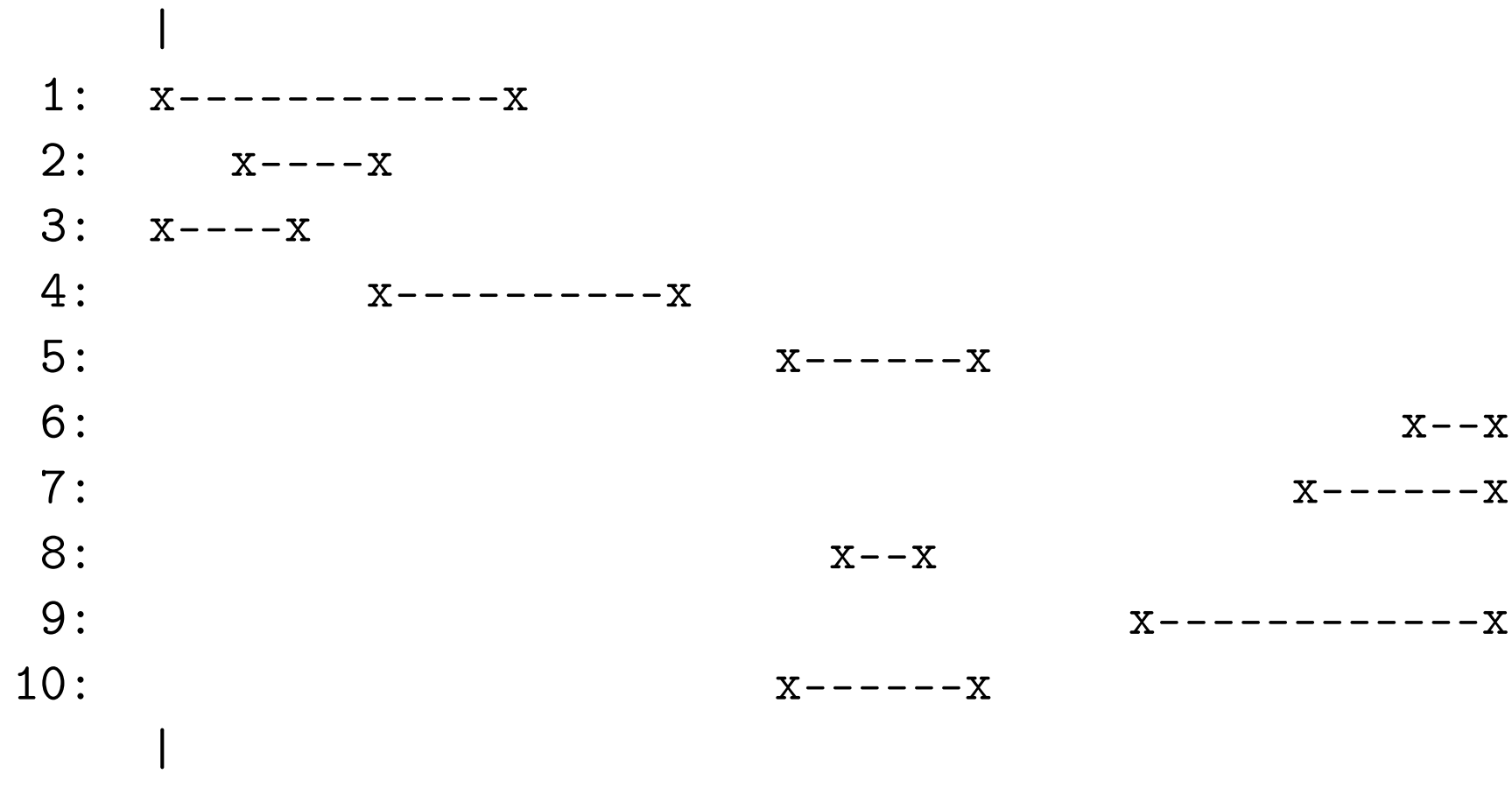
- Hver tala í rununni er sett einu sinni í hlaupabilið og mögulega fjarlægð úr því.
- Svo tímaflækjan er $\mathcal{O}(\, n \,)$.

- Þetta dæmi var í auðveldari kantinum.
- Skoðum annað dæmi:
- Byjrum á nokkrum undirstöðu atriðum.
- Tvö bil kallast *næstum sundurlæg* ef sniðmengi þeirra er tómt eða bara einn punktur.
- Sammengi bila má skrifa sem sammengi næstu sundurlægra bila.
- *Lengd bilsins* $[a, b]$ er $b - a$.
- Til að finna *lengd sammengis bila* skrifum við sammengið sem sammengi næstum sundurlægra bila og tökum summu lengda þeirra.
- Til dæmis eru bilin $[1, 2]$ og $[2, 3]$ næstum sundurlæg (en þó ekki sundurlæg) en $[1, 3]$ og $[2, 4]$ eru það ekki. Nú $[1, 3] \cup [2, 4] = [1, 4]$ svo lengd $[1, 3] \cup [2, 4]$ er 3.

- Gefið n bil hver er lengd sammengis þeirra.

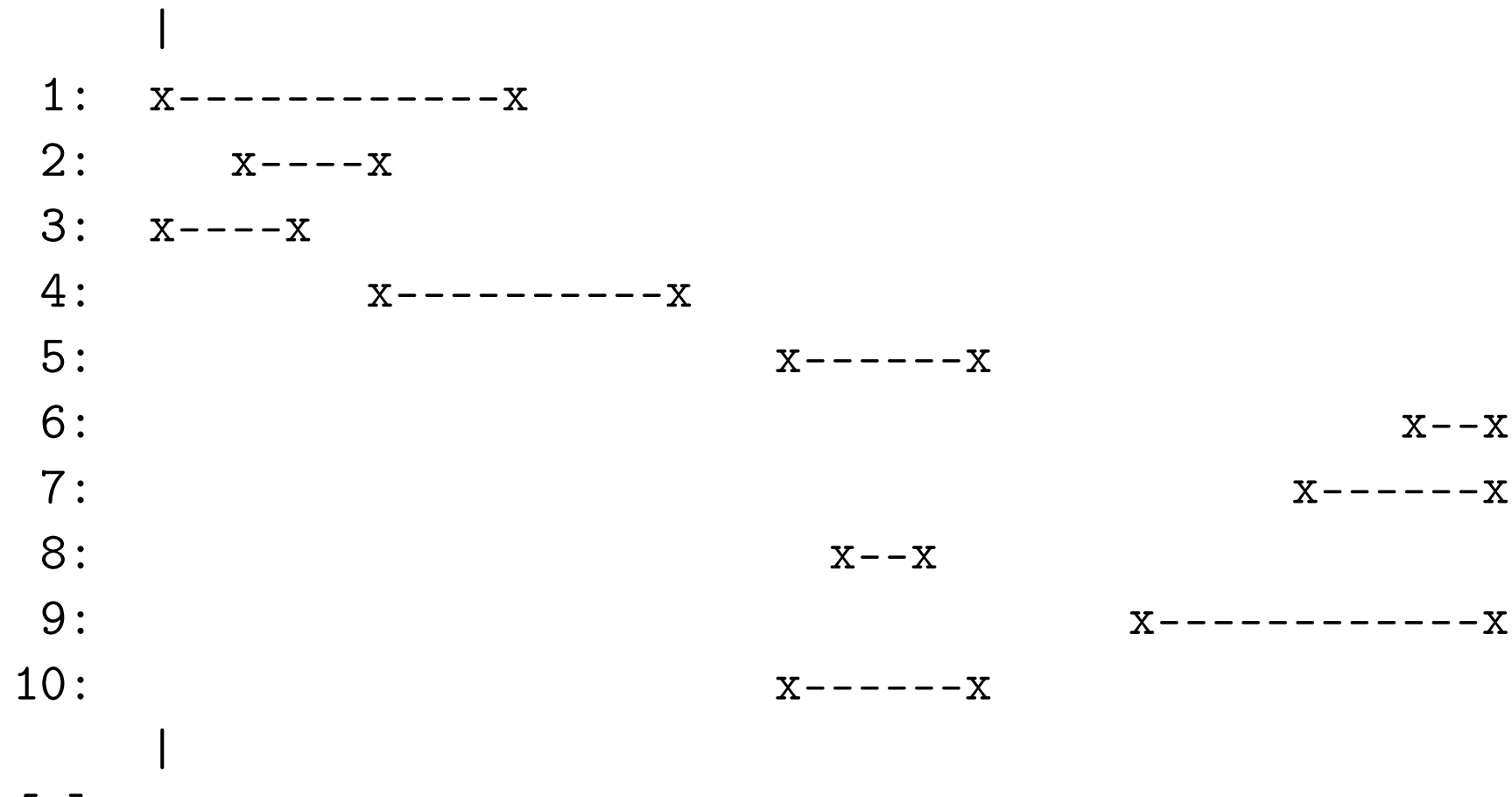
- Geymum í lista tvenndir þar sem fyrra stakið er endapunktur bils og seinna stakið segir hvaða bili punkturinn tilleyrir.
- Röðum þessum punktum svo í vaxandi röð.
- Við löbbum í gegnum þennan raðaða lista og höldum utan um hlaupabil þannig að við bætum við bili í hlaupabilið þegar við rekumst á vinstri endapunkt þess og fjarlægjum það þegar við rekumst á hægri endapunkt þess.
- Við skoðum svo sérstaklega tilfellin þegar við erum ekki með nein bil í hlaupabilinu okkar.
- Sammengi þeirra bila sem við höfum farið í gegnum þá síðan hlaupabilið var síðast tómt er nú næstum sundurlægt öllum öðrum bilum sem okkur var gefið í byrjun.
- Við skilum því summu lengda þessara sammengja.

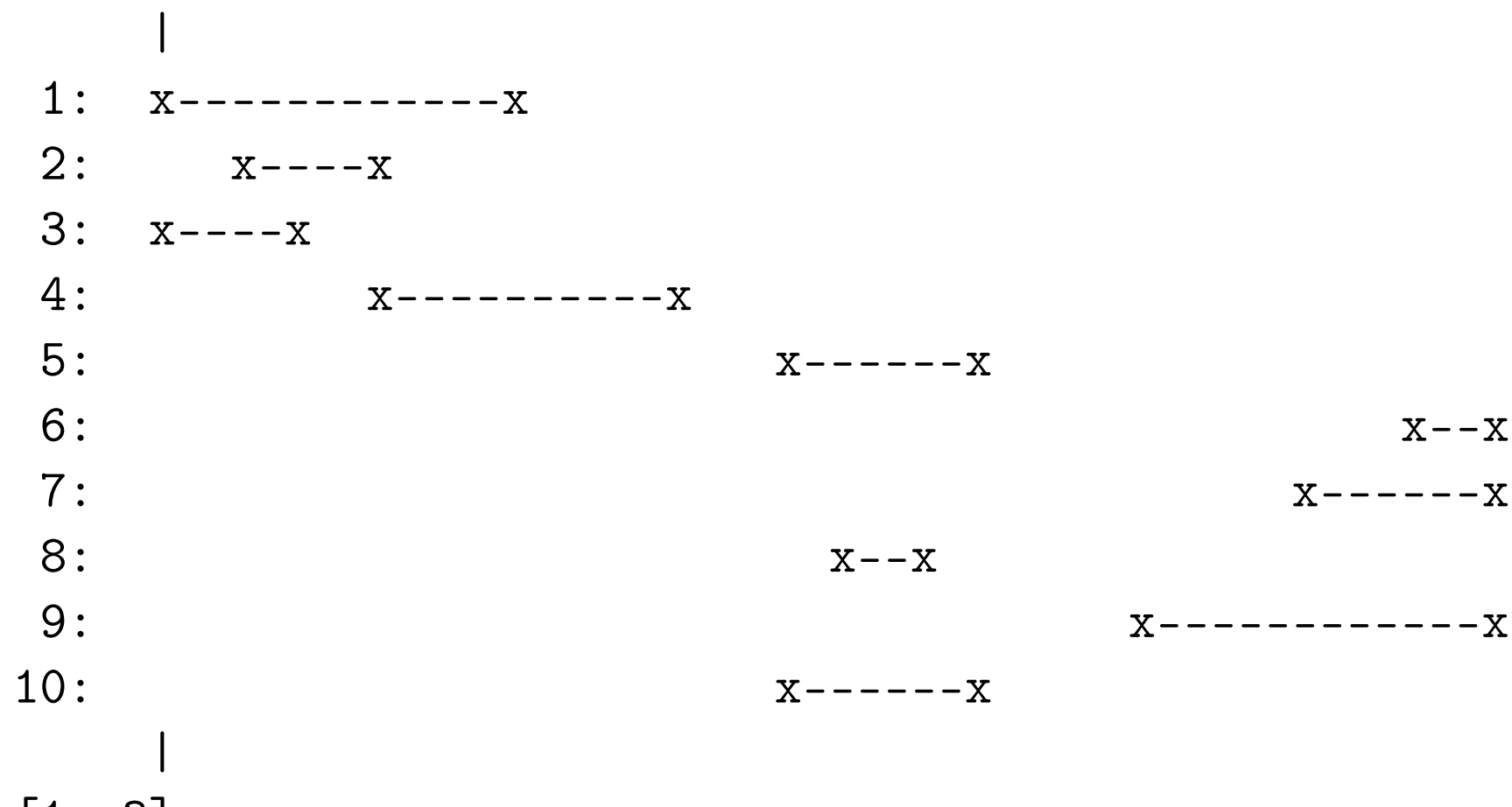

1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[]
r = 0



[]

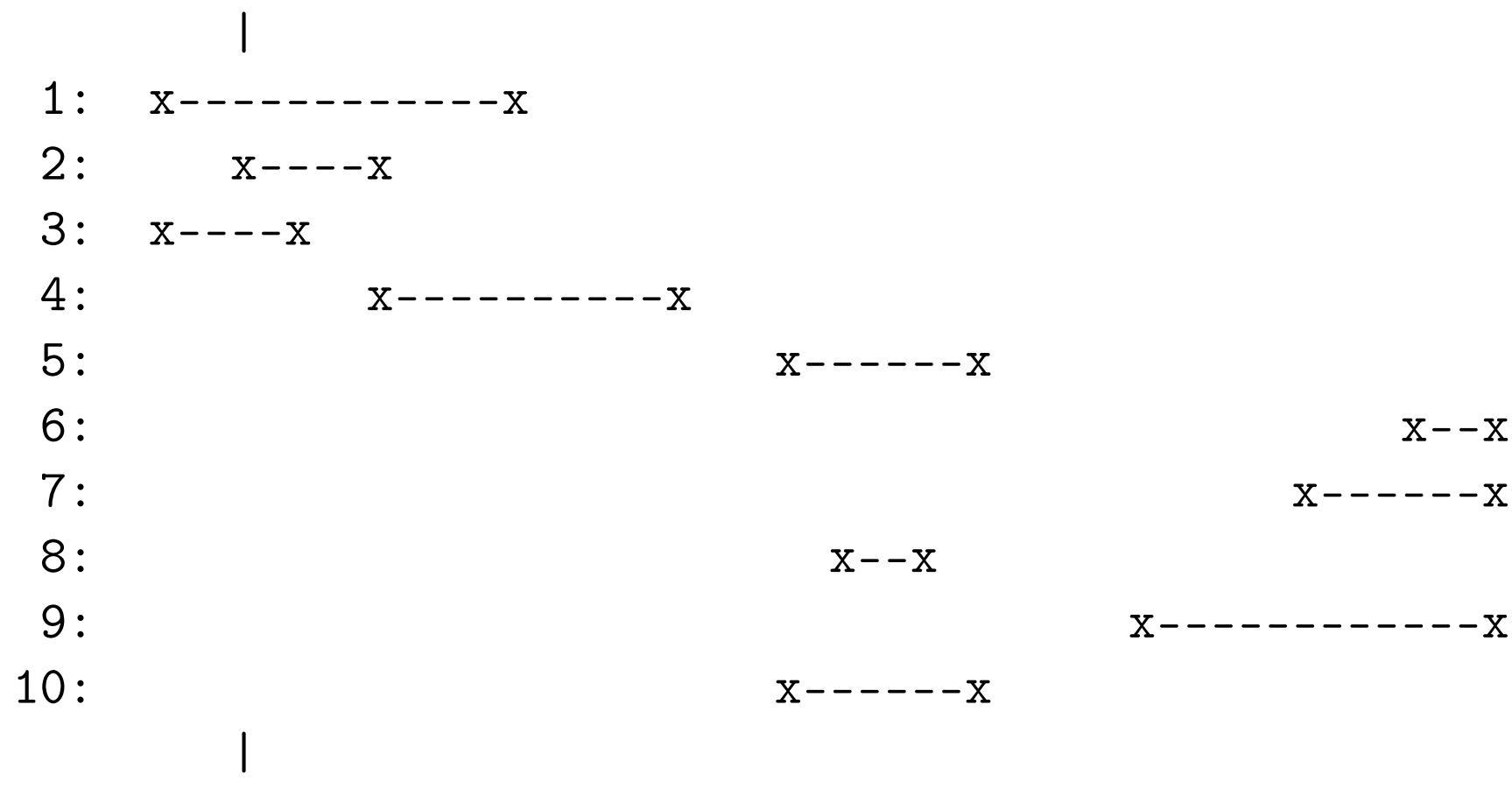
r = 0


1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[1]
r = 0



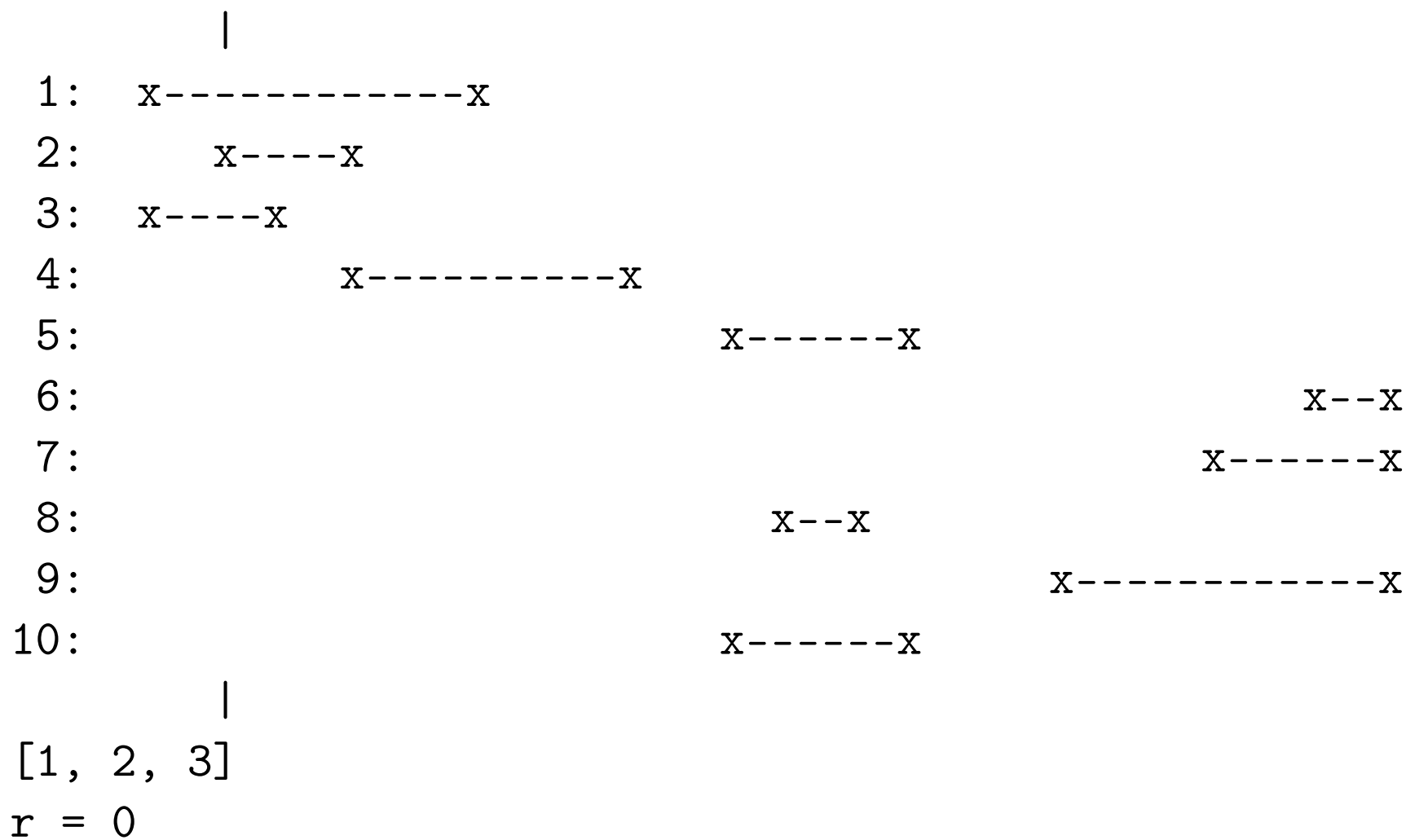
[1, 3]

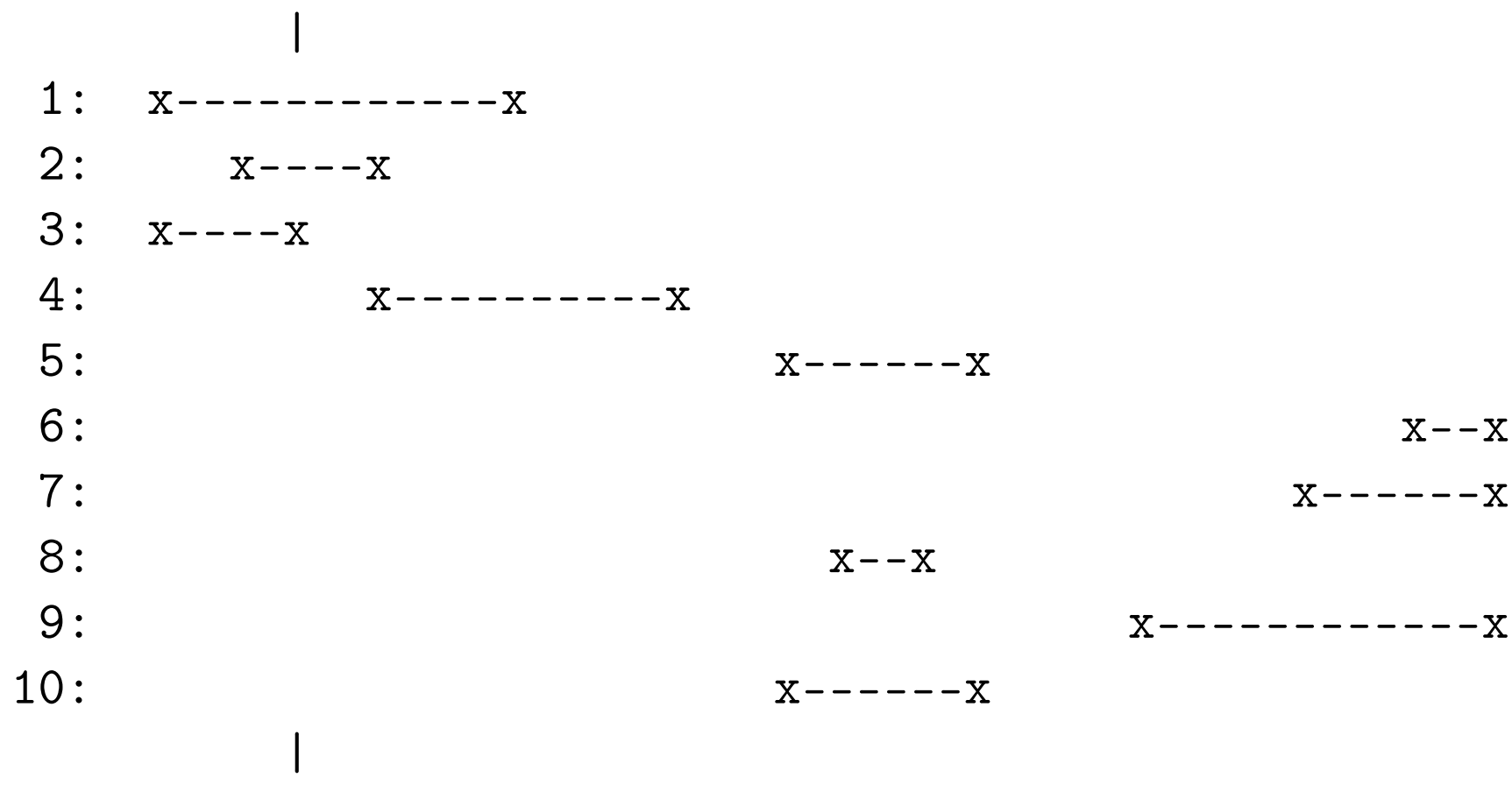
r = 0



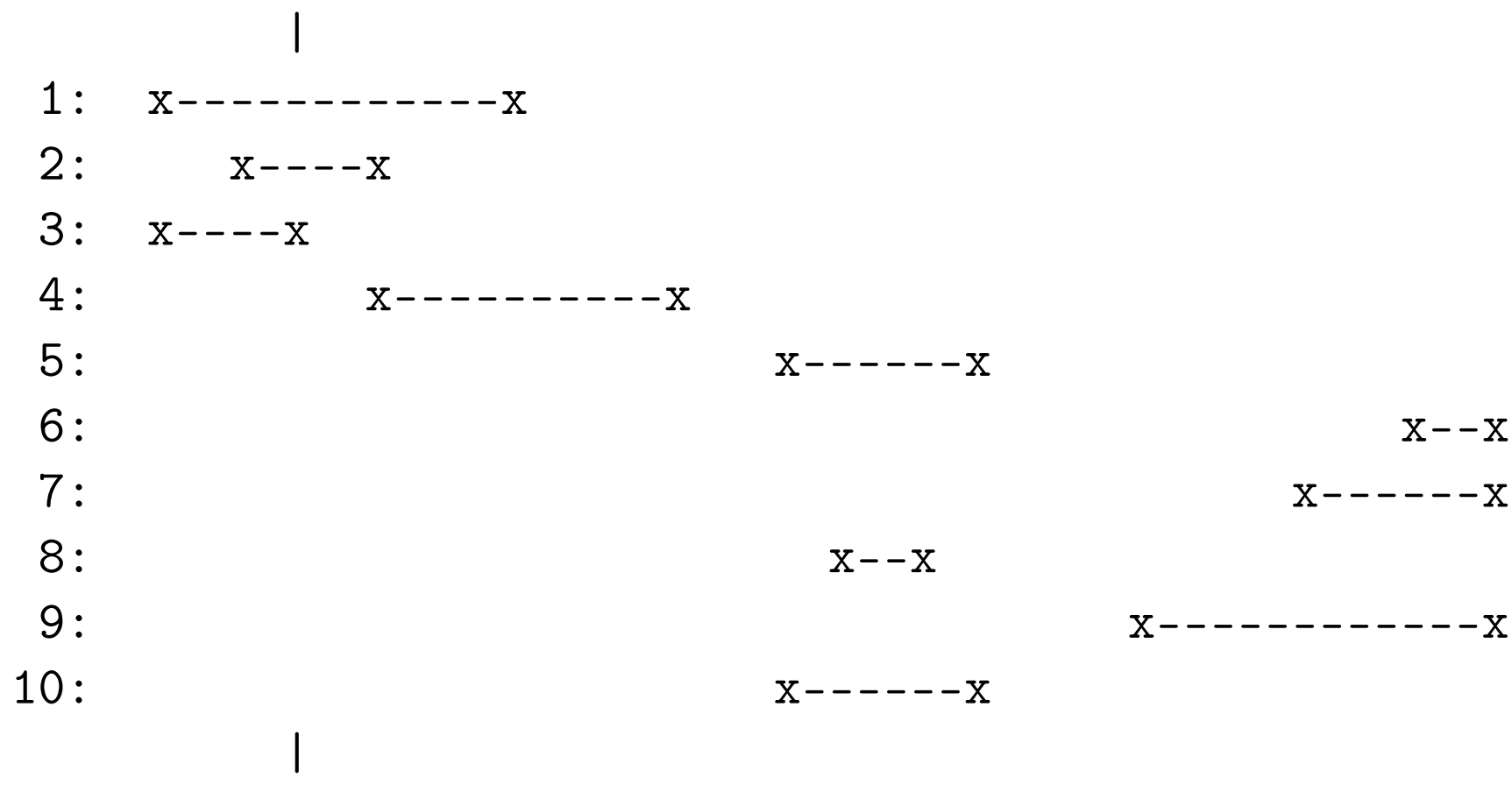
[1, 3]

r = 0


1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[1, 2, 3]
r = 0

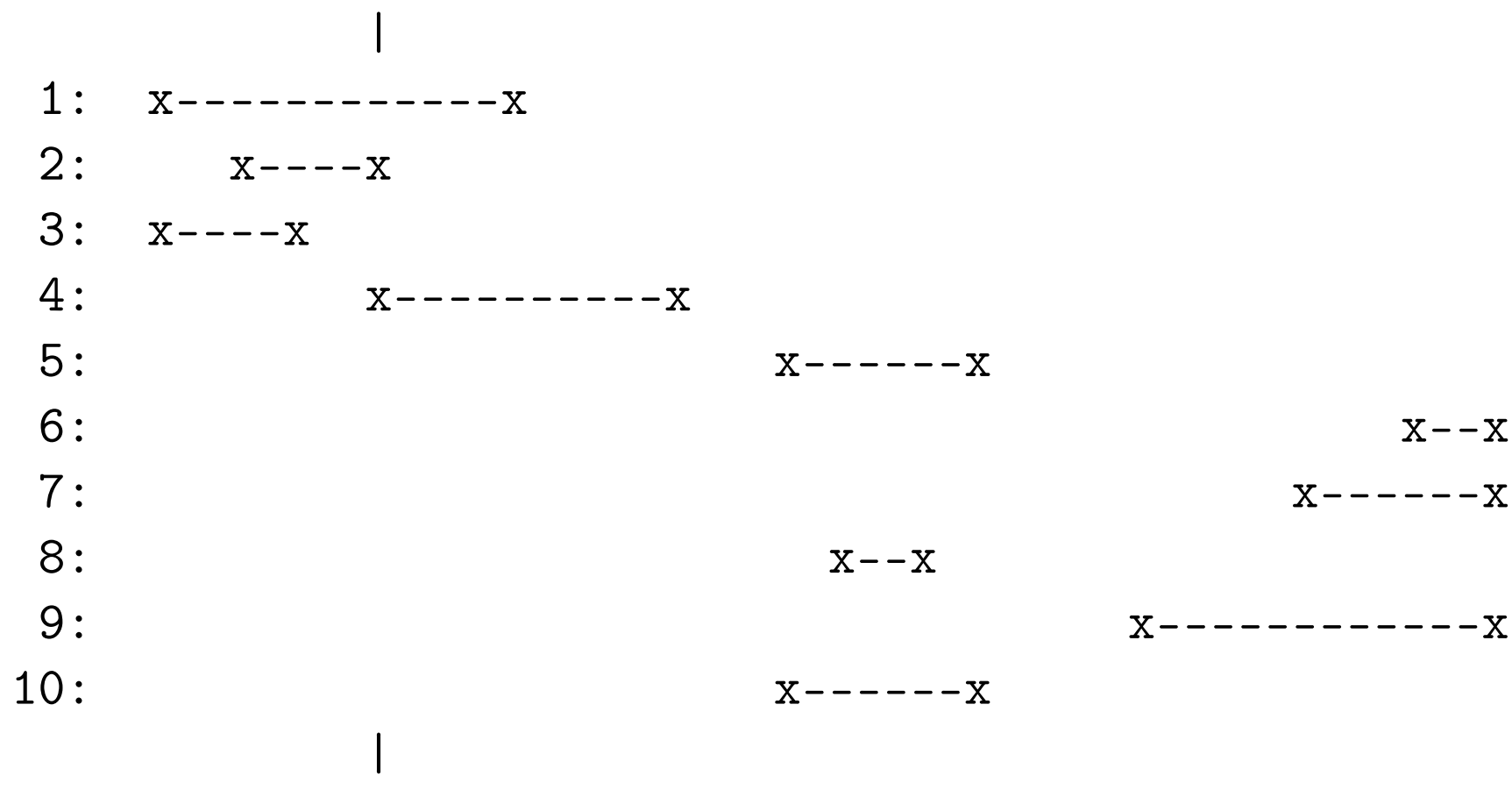


[1, 2, 3]
r = 0



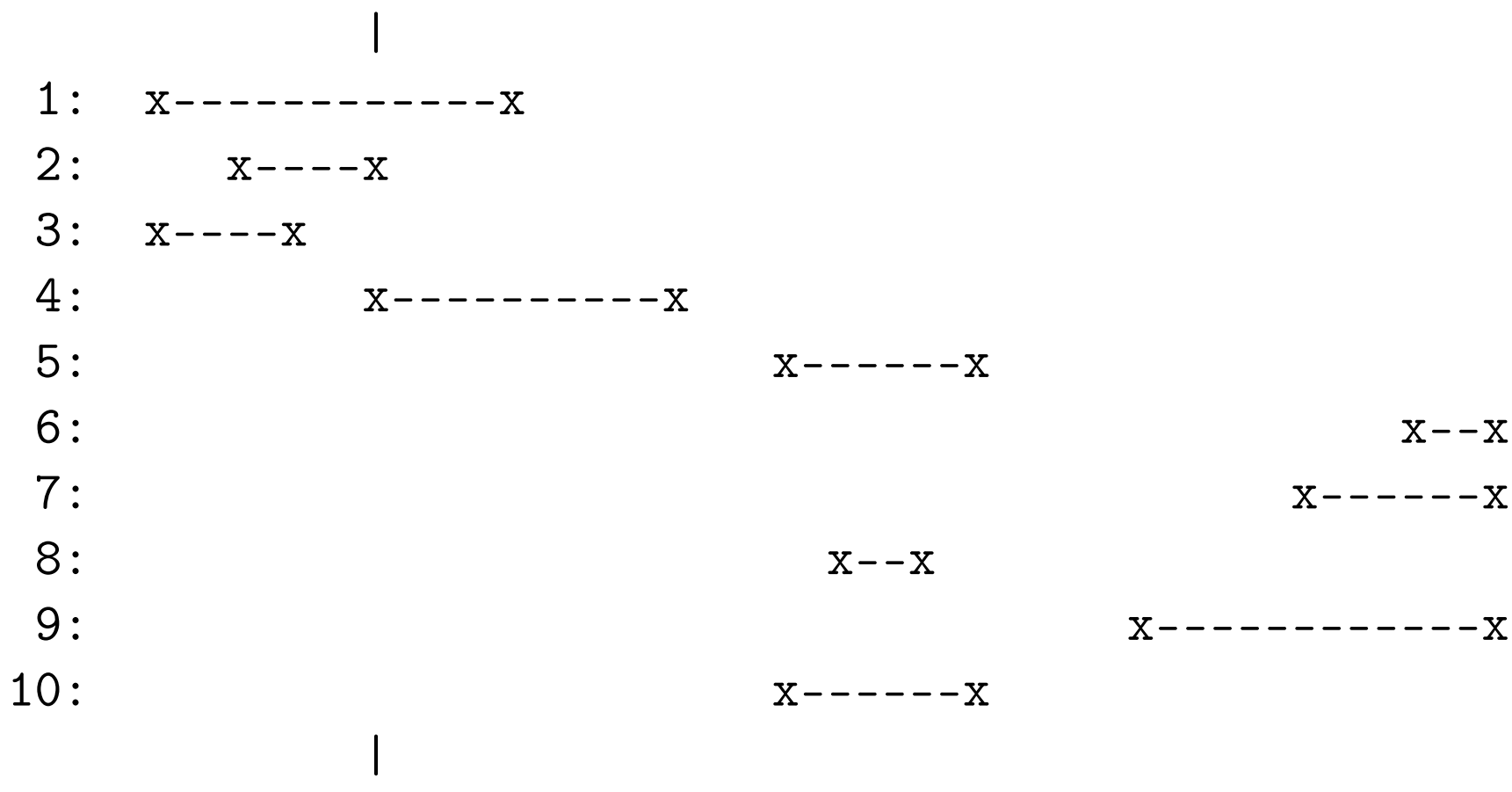
[1, 2]

r = 0



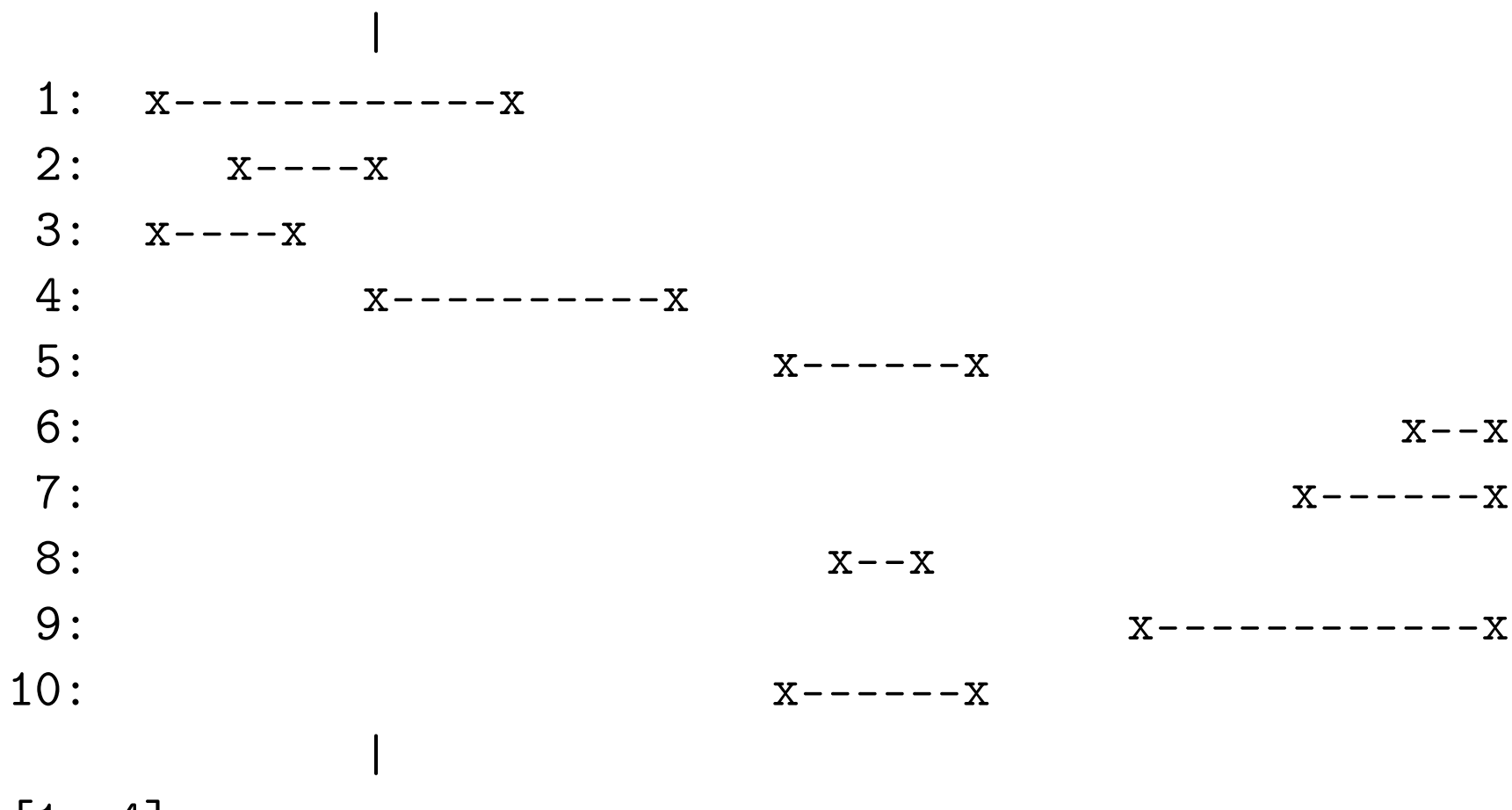
[1, 2]

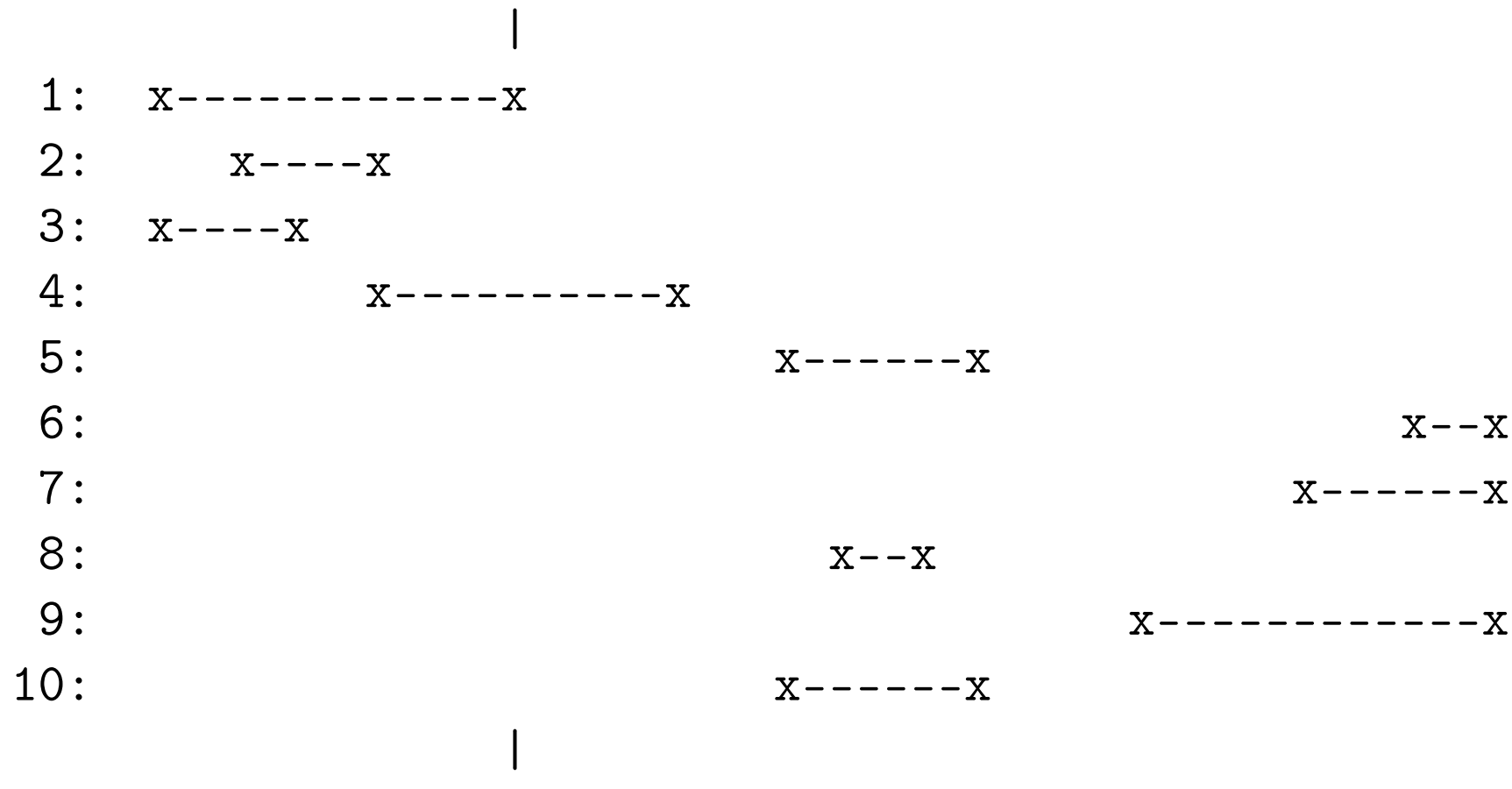
r = 0



[1, 2, 4]

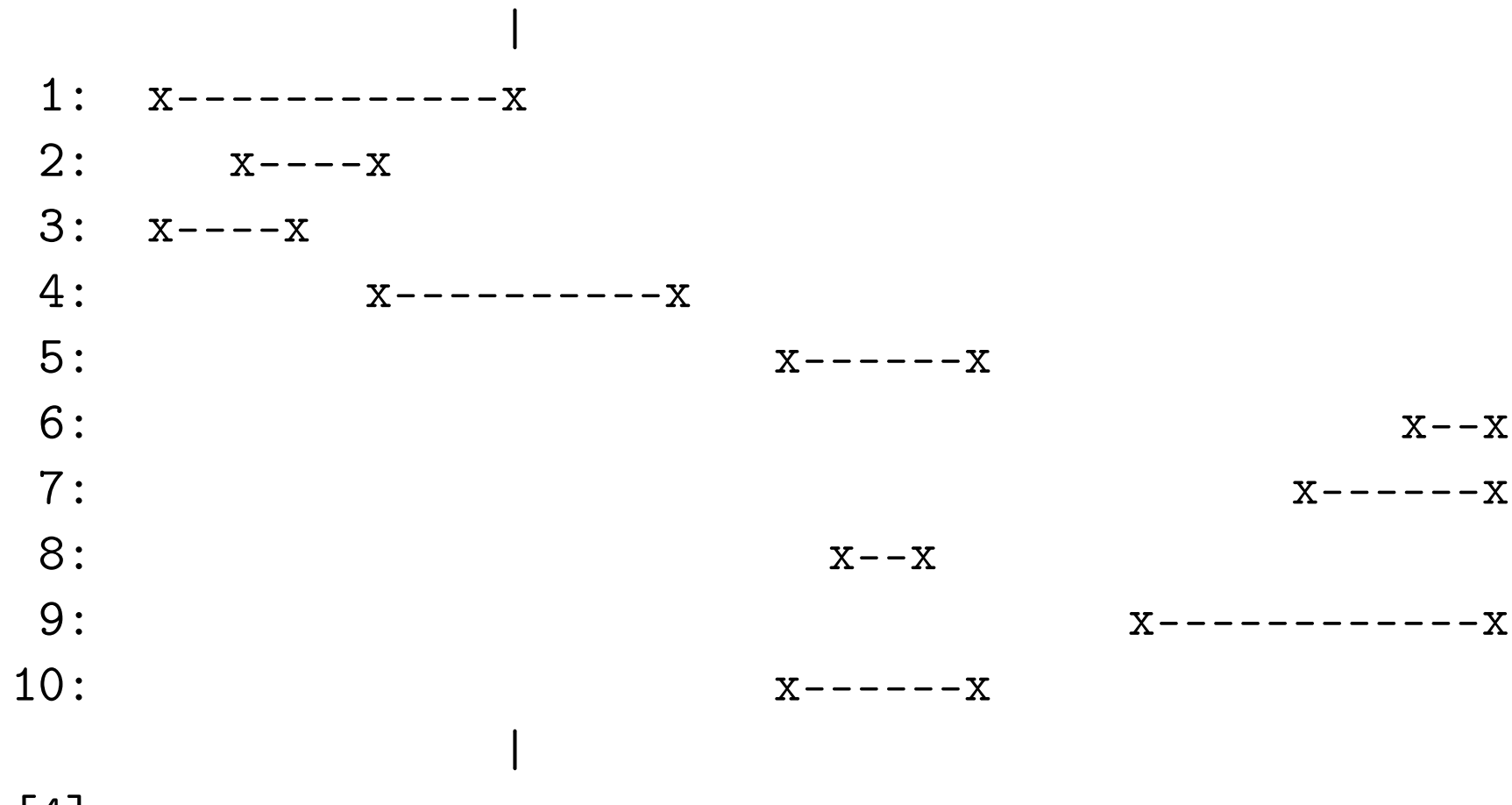
r = 0


1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[1, 4]
r = 0



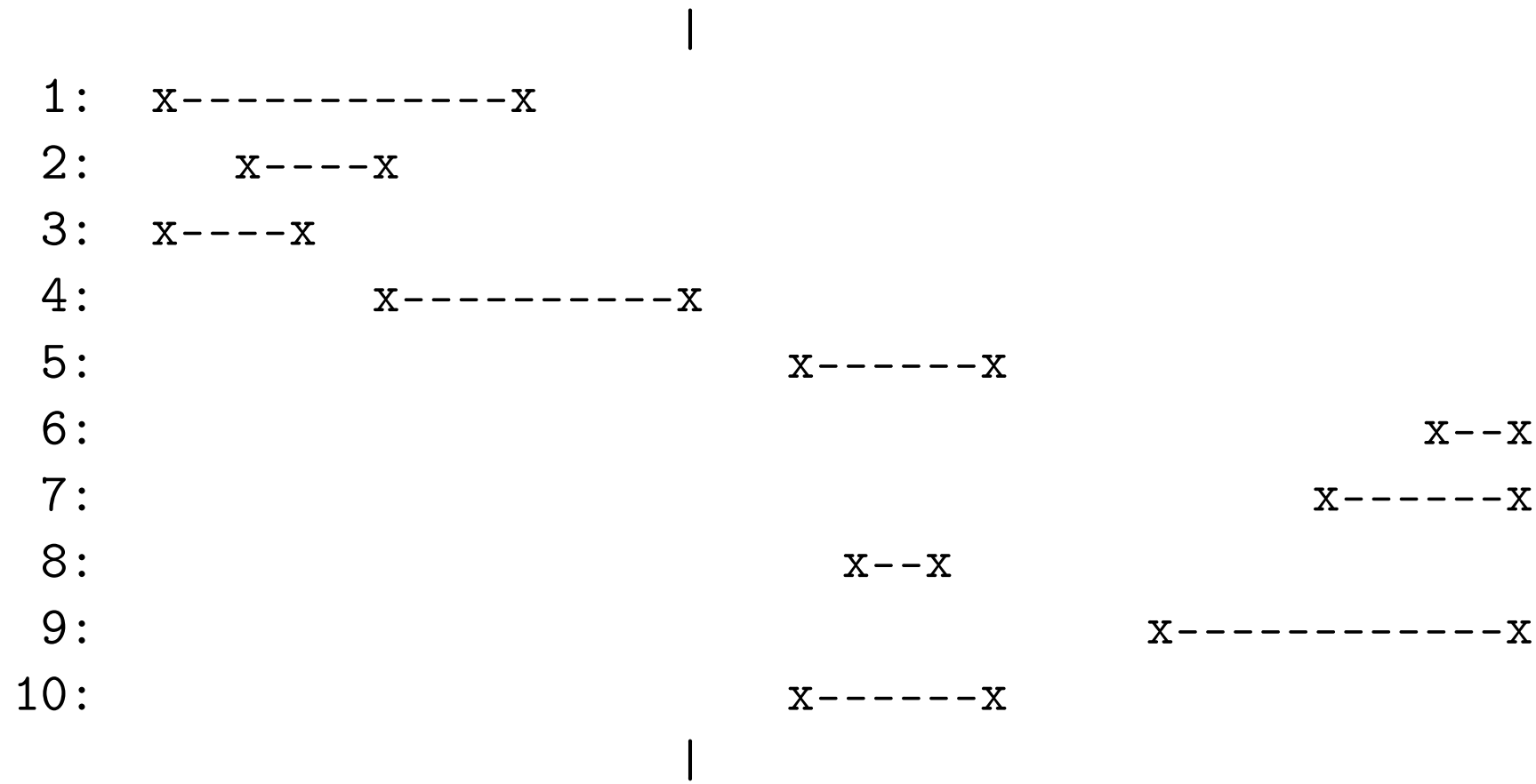
[1, 4]

r = 0



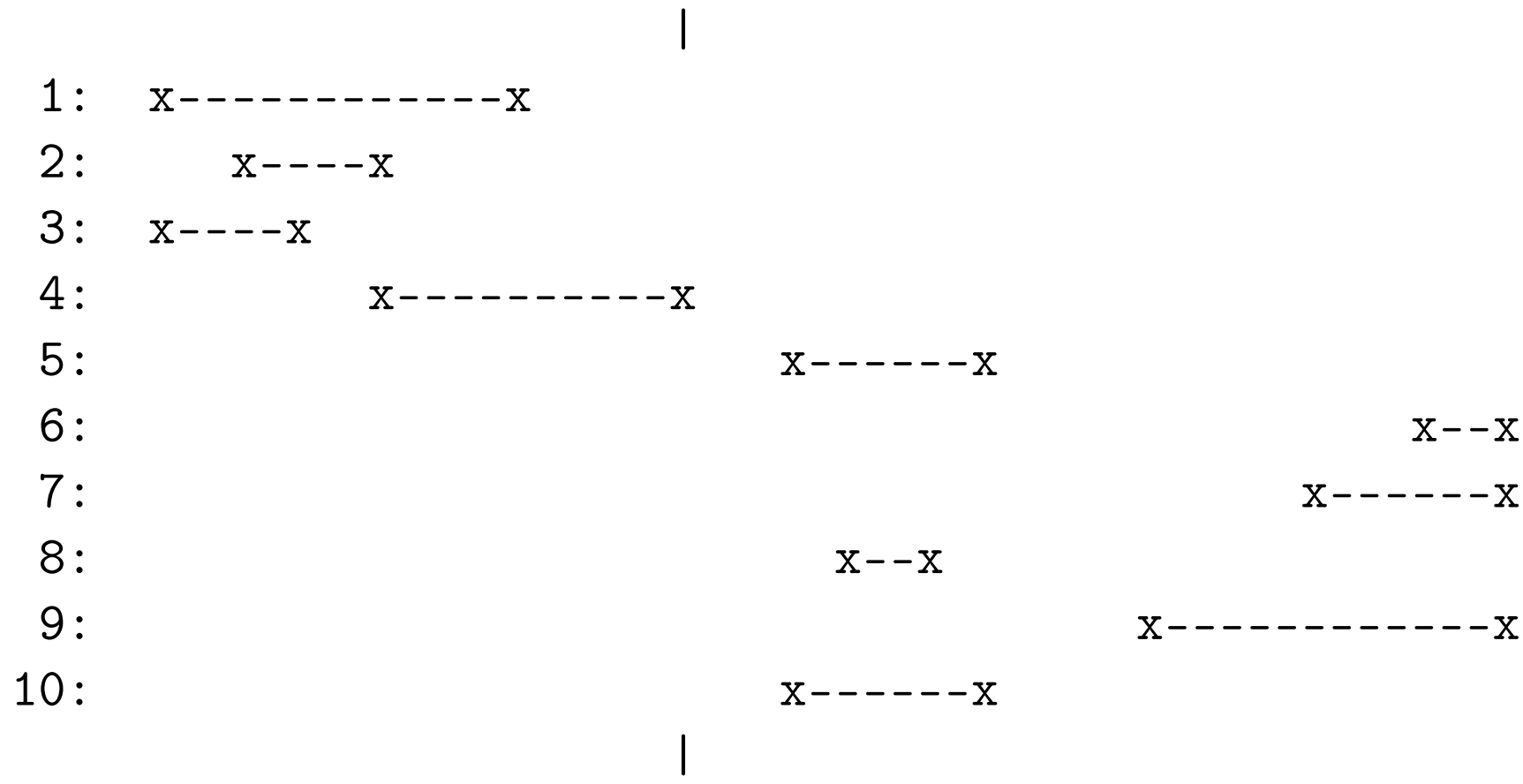
[4]

r = 0



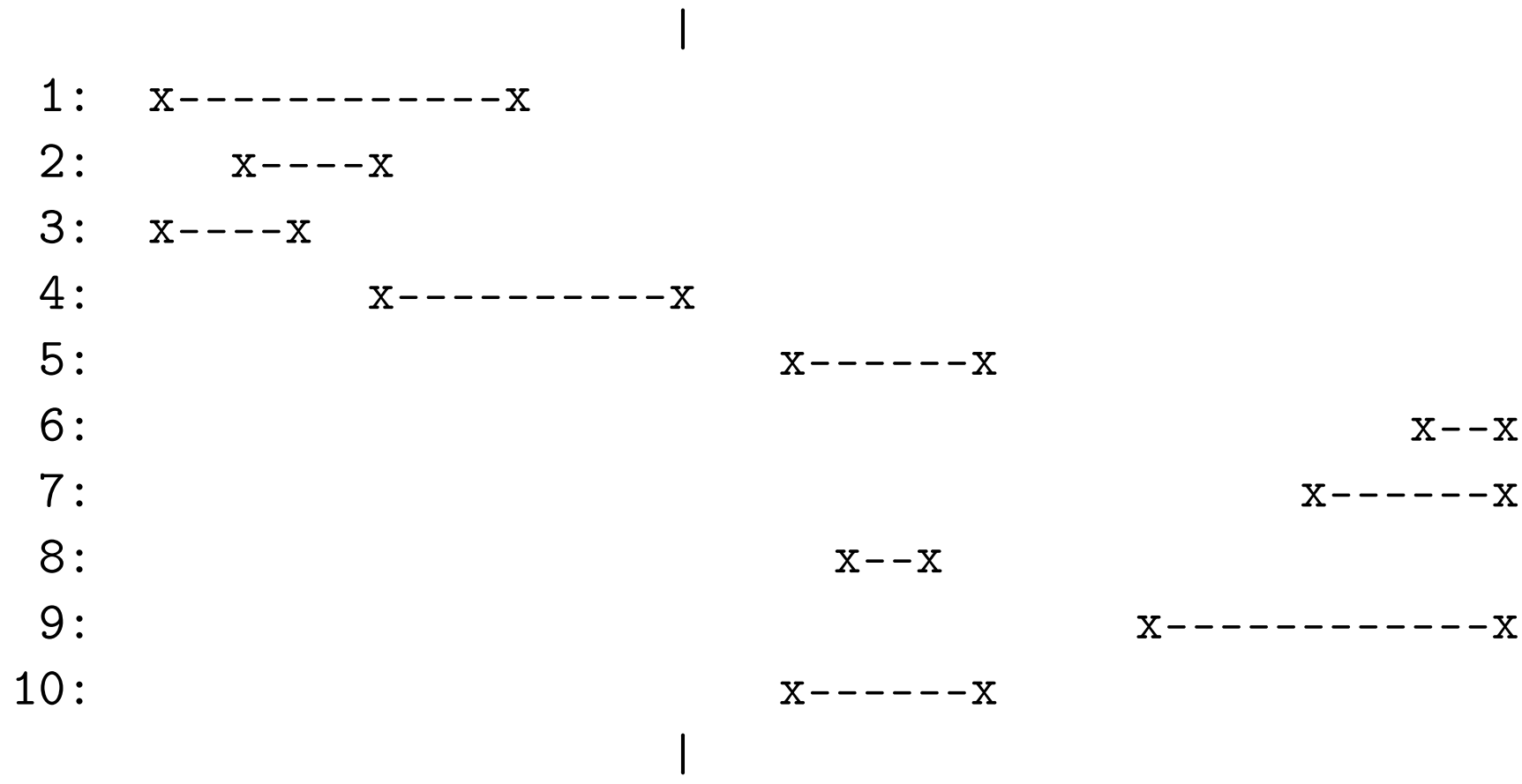
[4]

r = 0



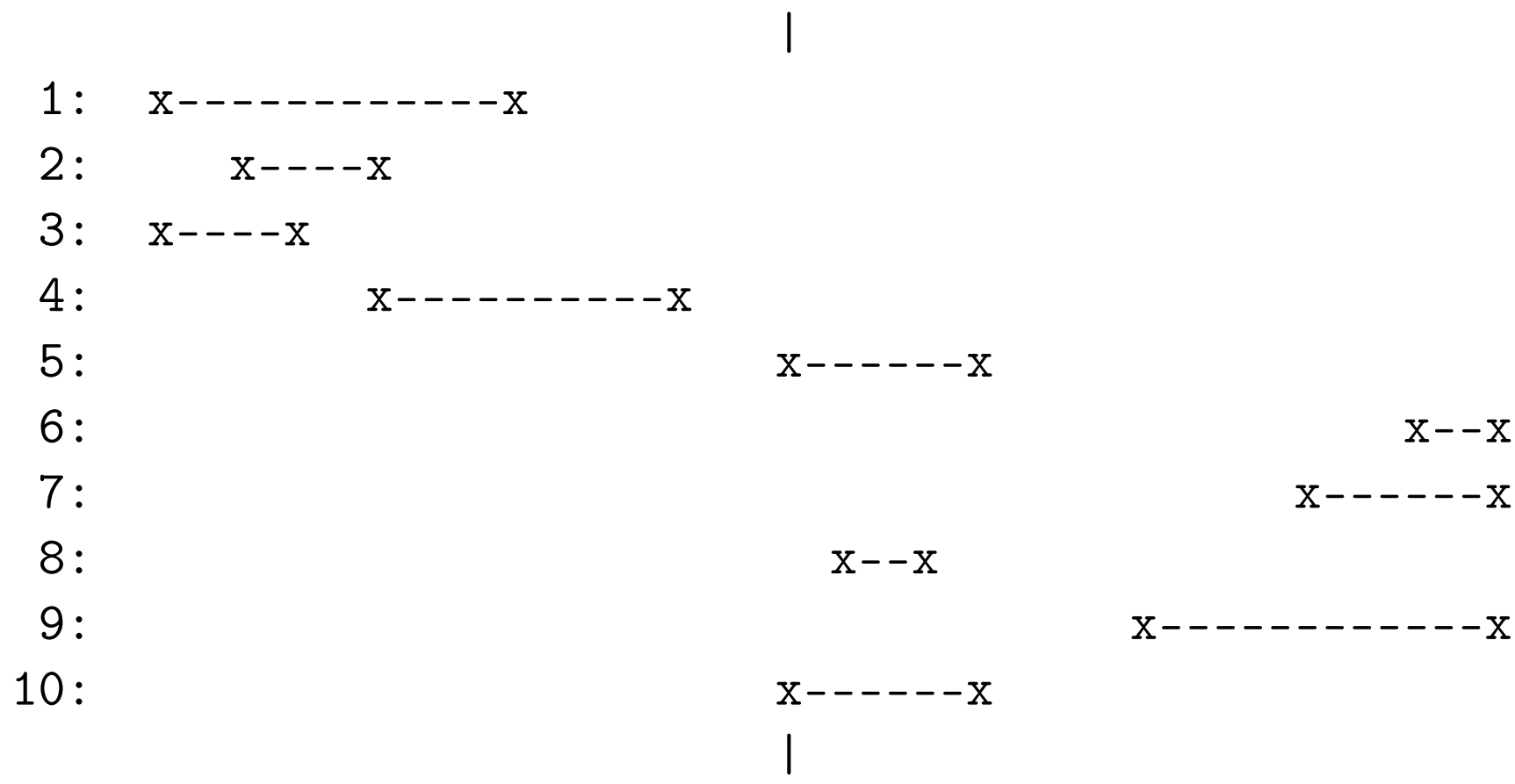
[]

r = 0



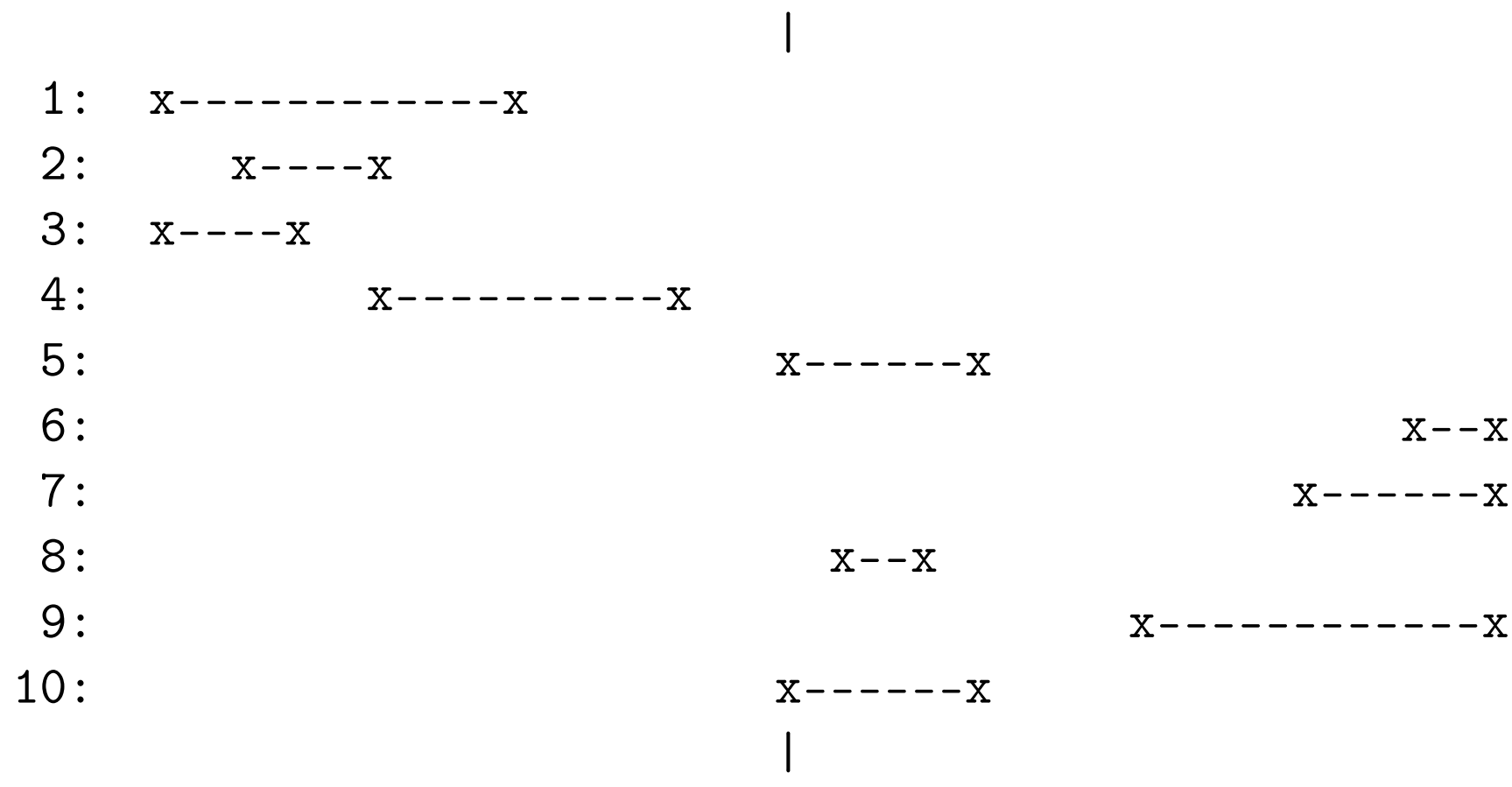
[]

r = 20



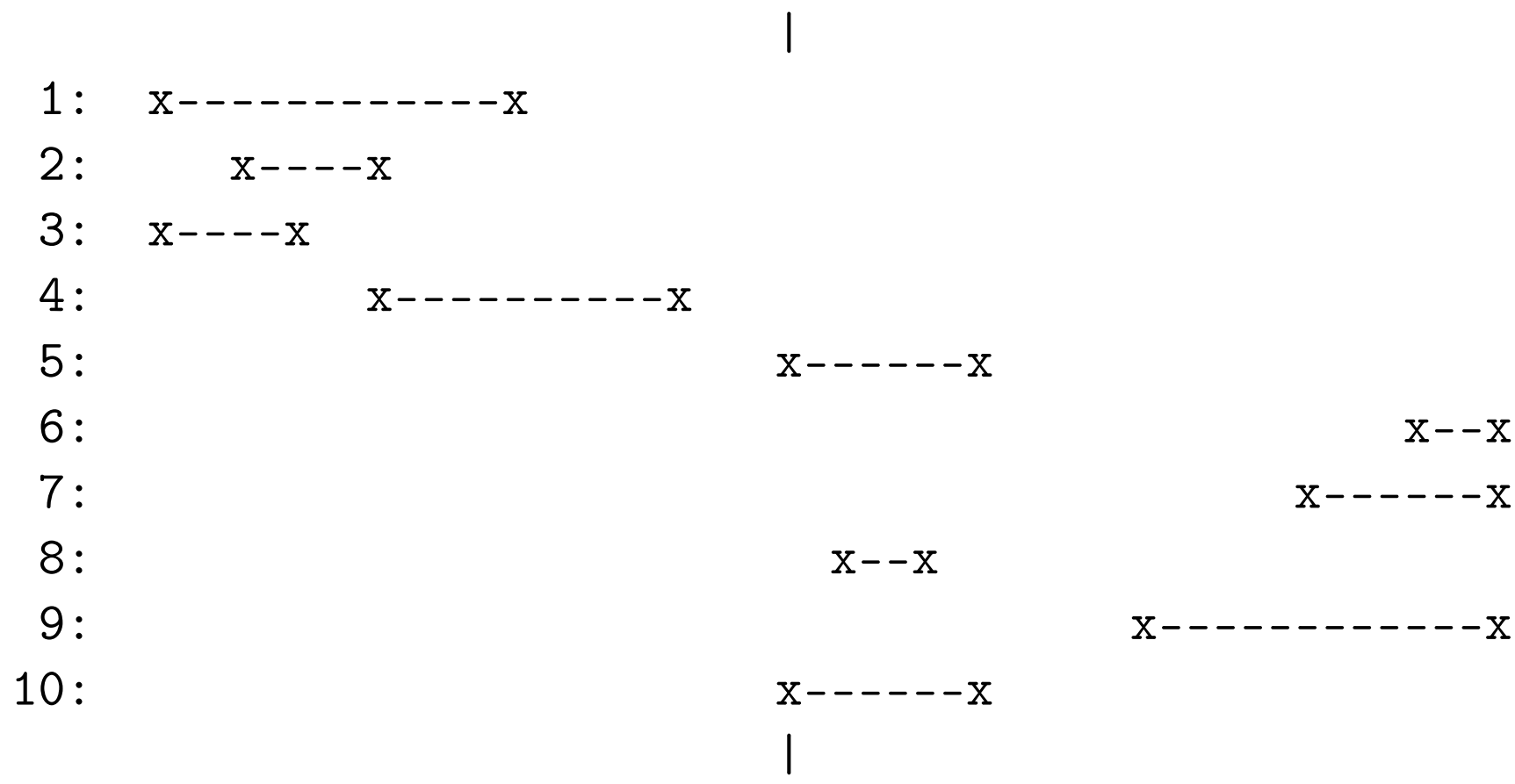
[]

r = 20

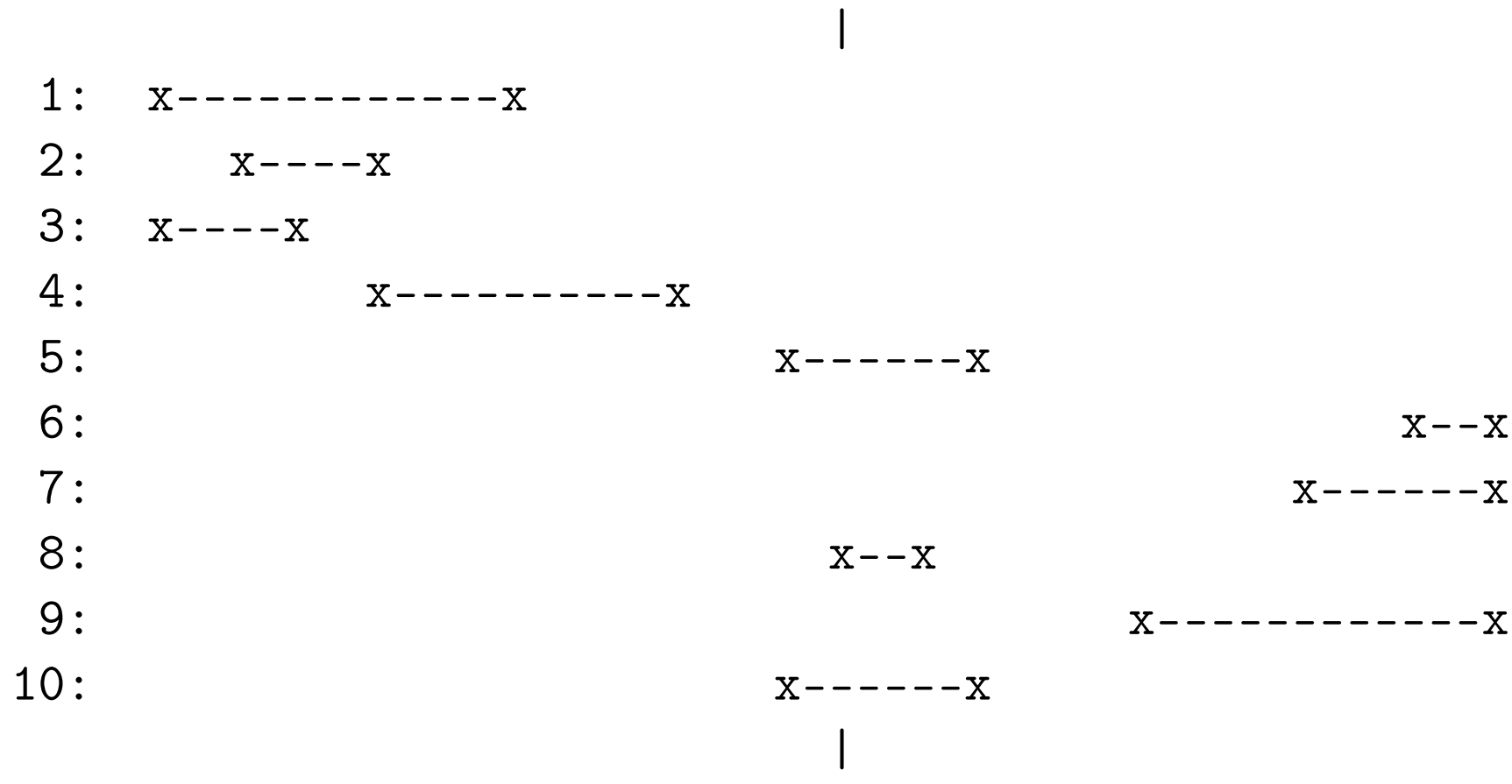


[5]

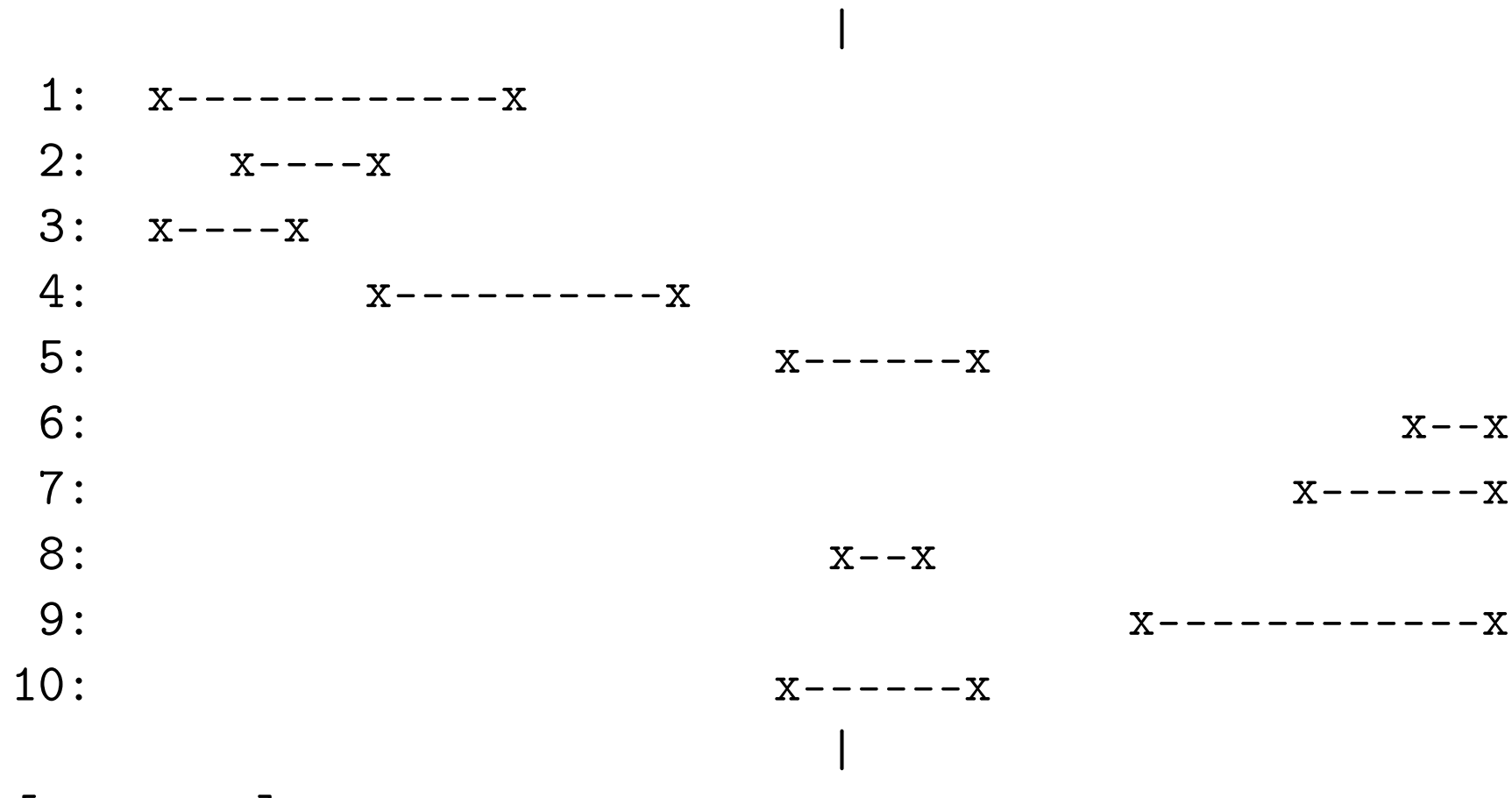
r = 20



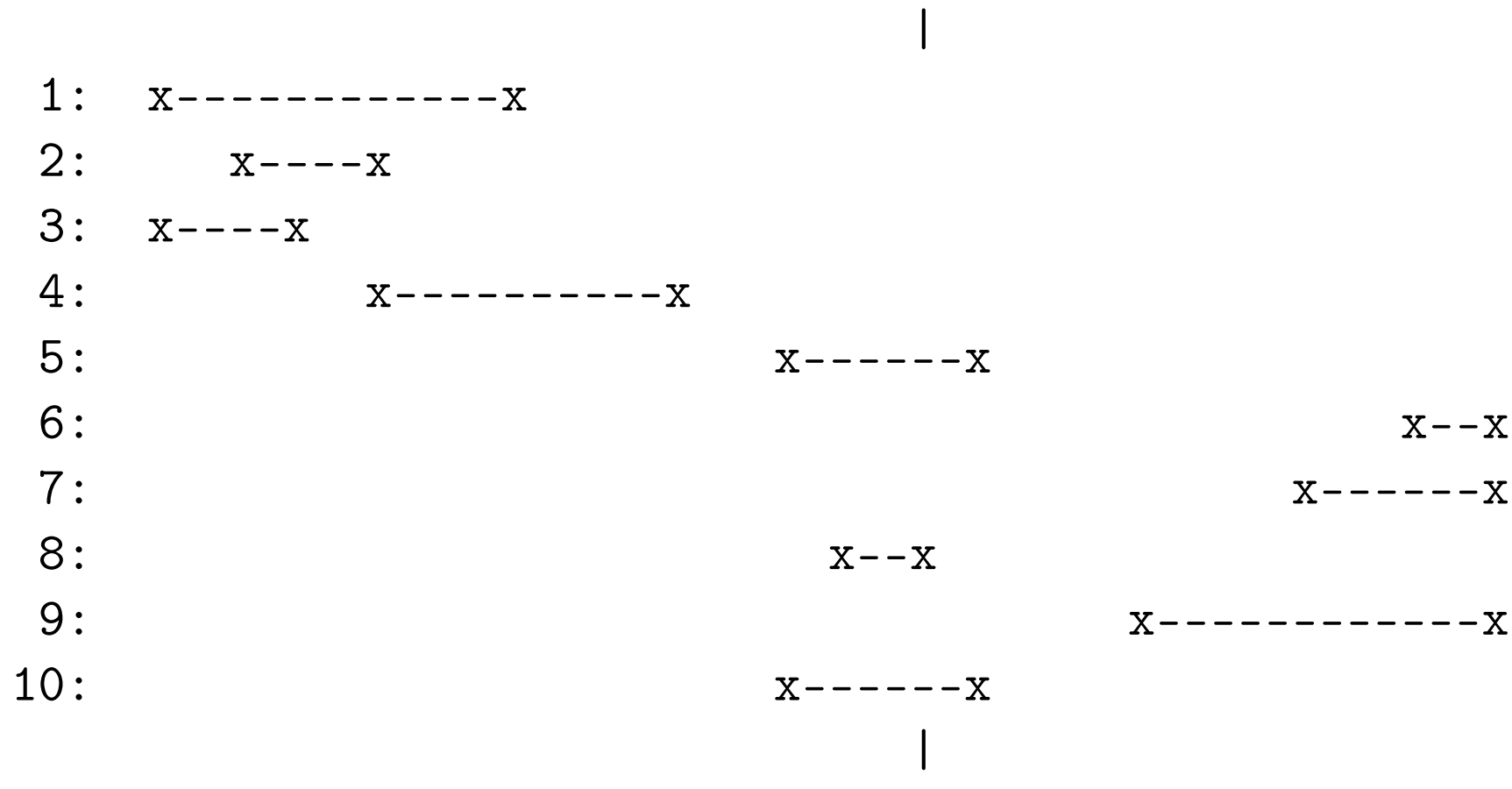
[5, 10]
r = 20



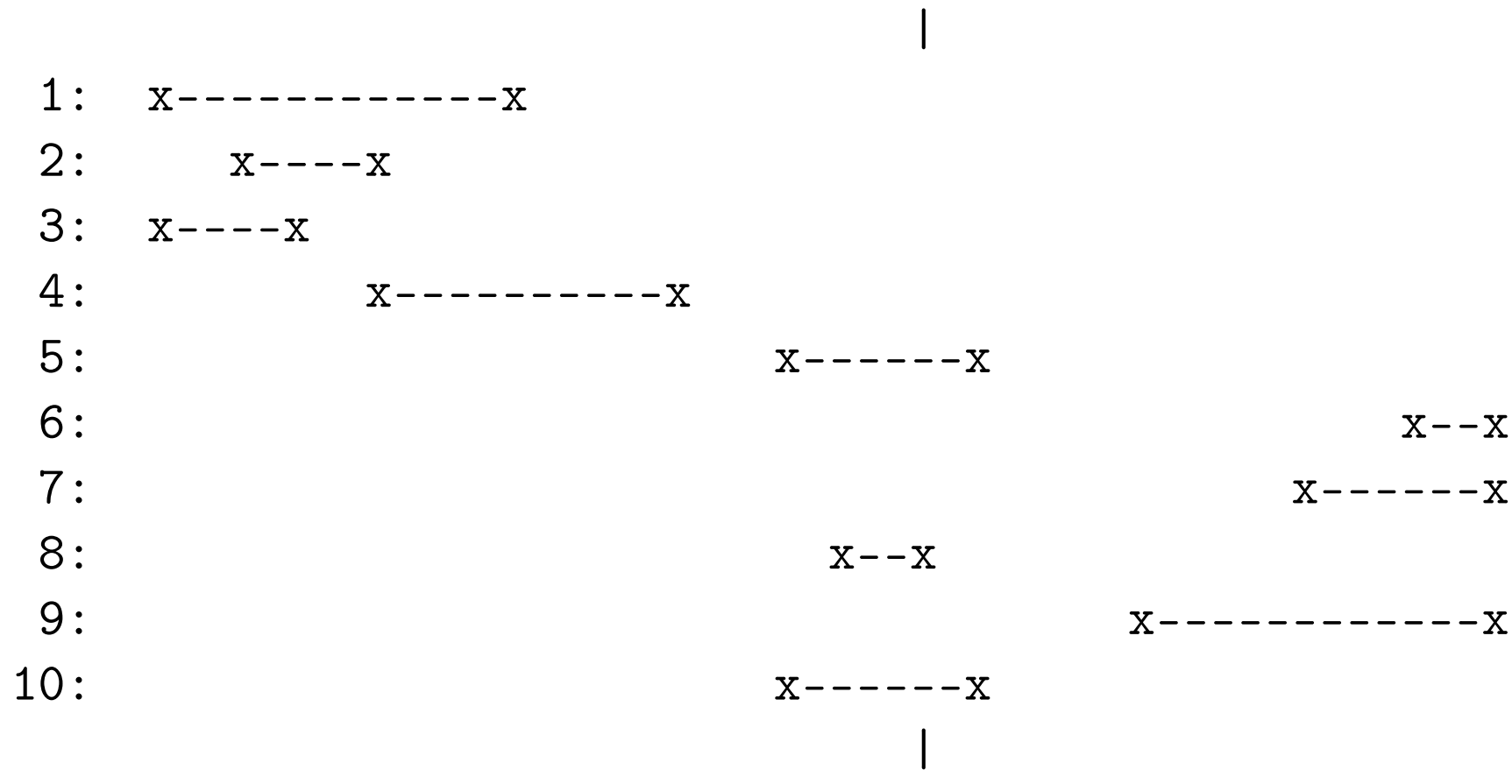
[5, 10]
r = 20



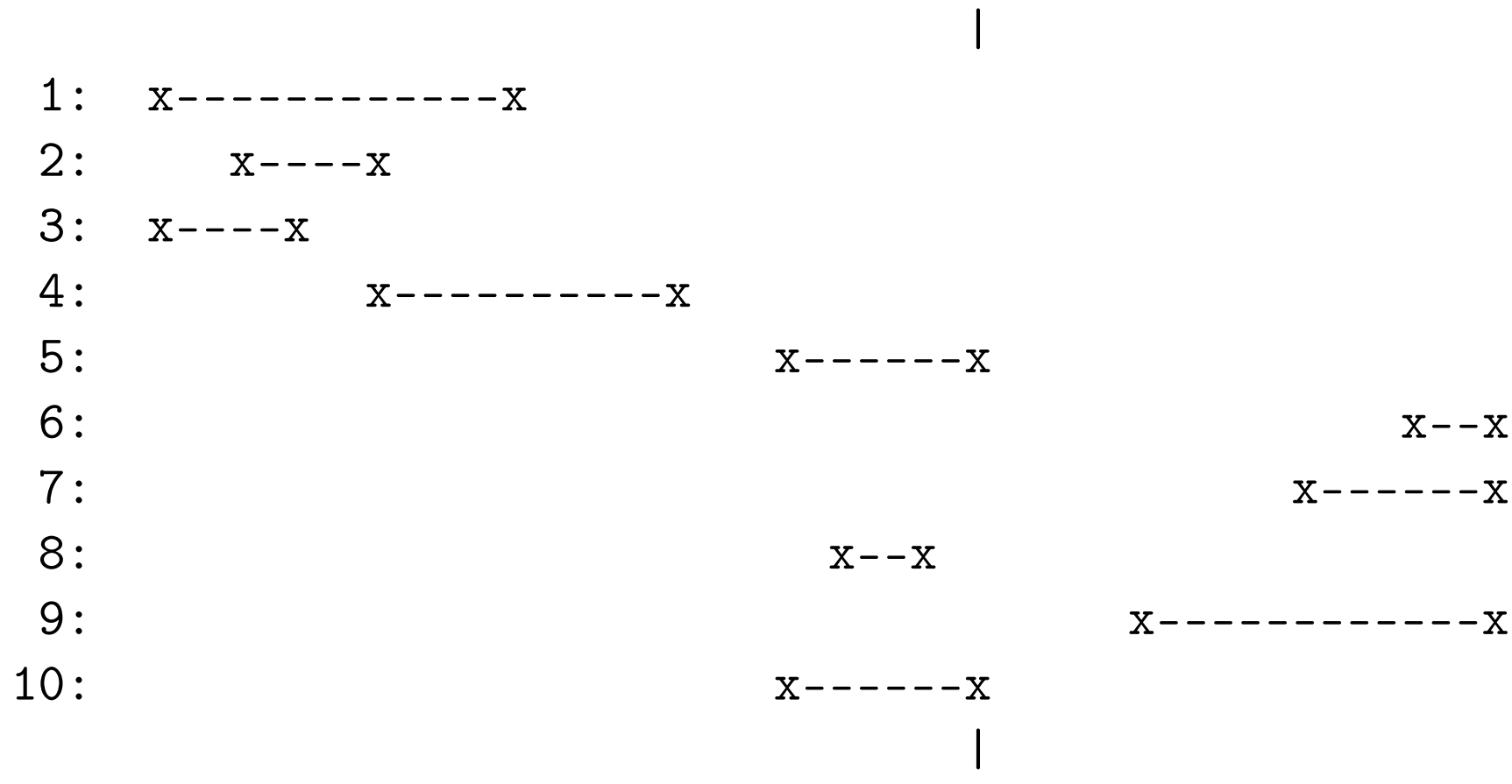
[5, 8, 10]
r = 20



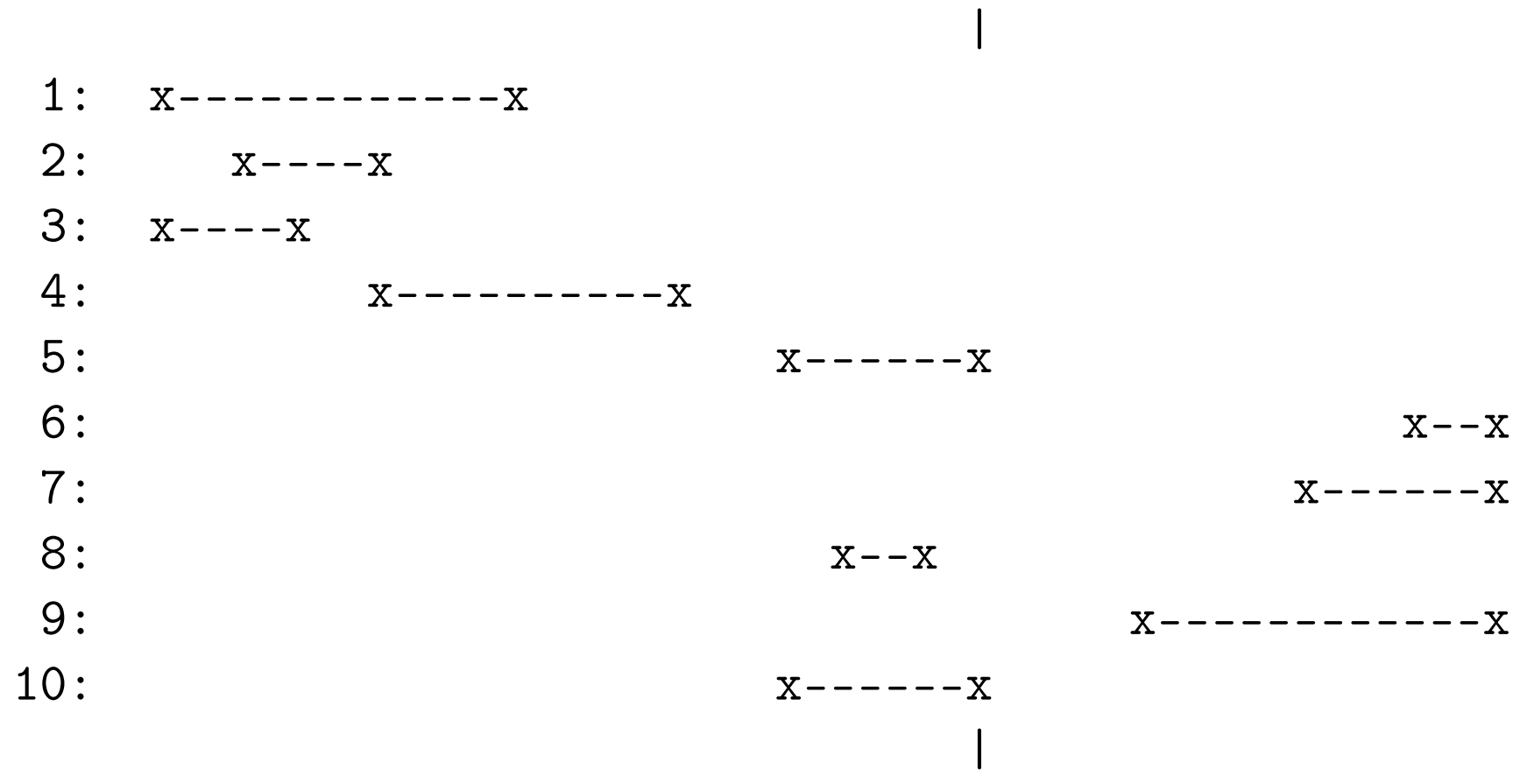
[5, 8, 10]
r = 20



[5, 10]
r = 20

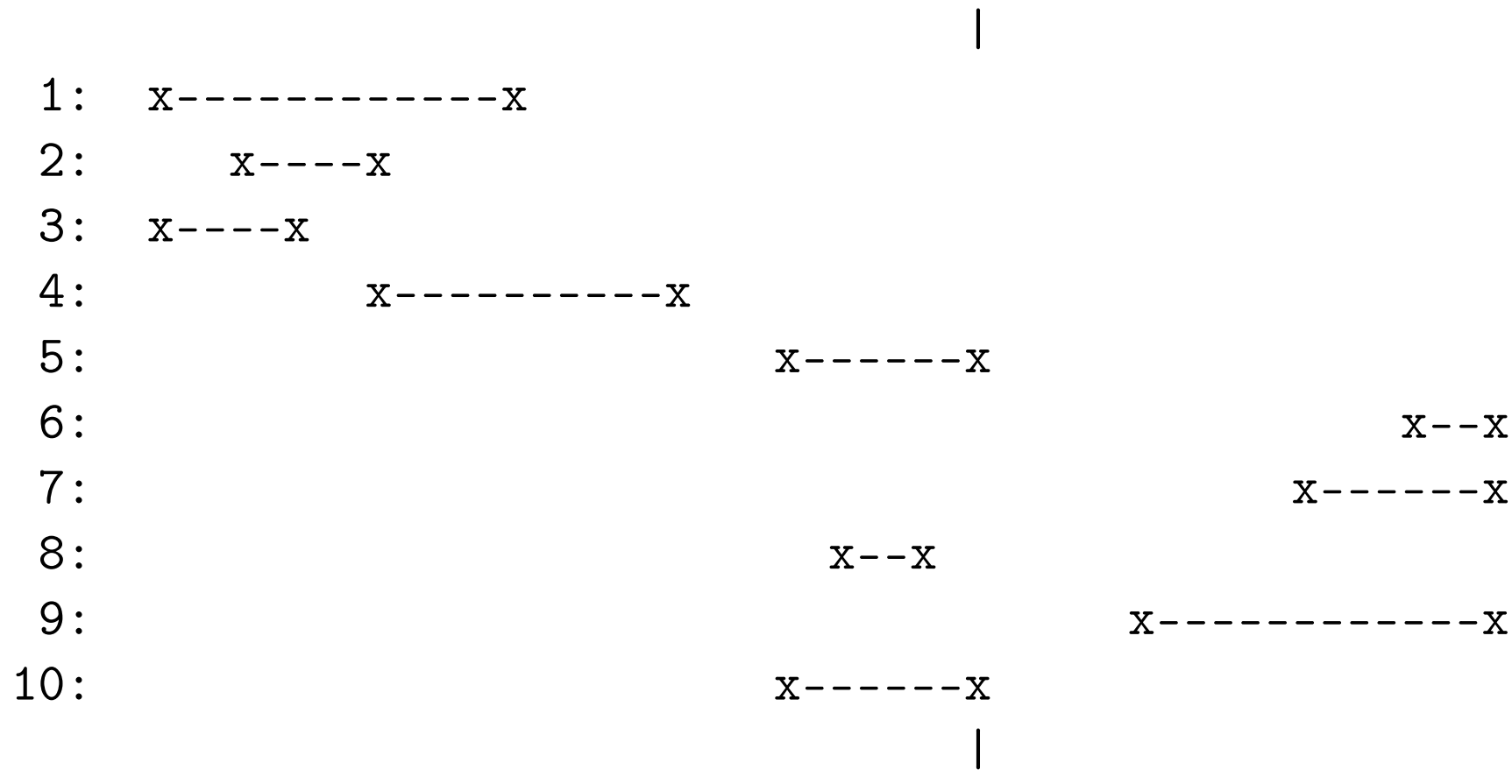


[5, 10]
r = 20



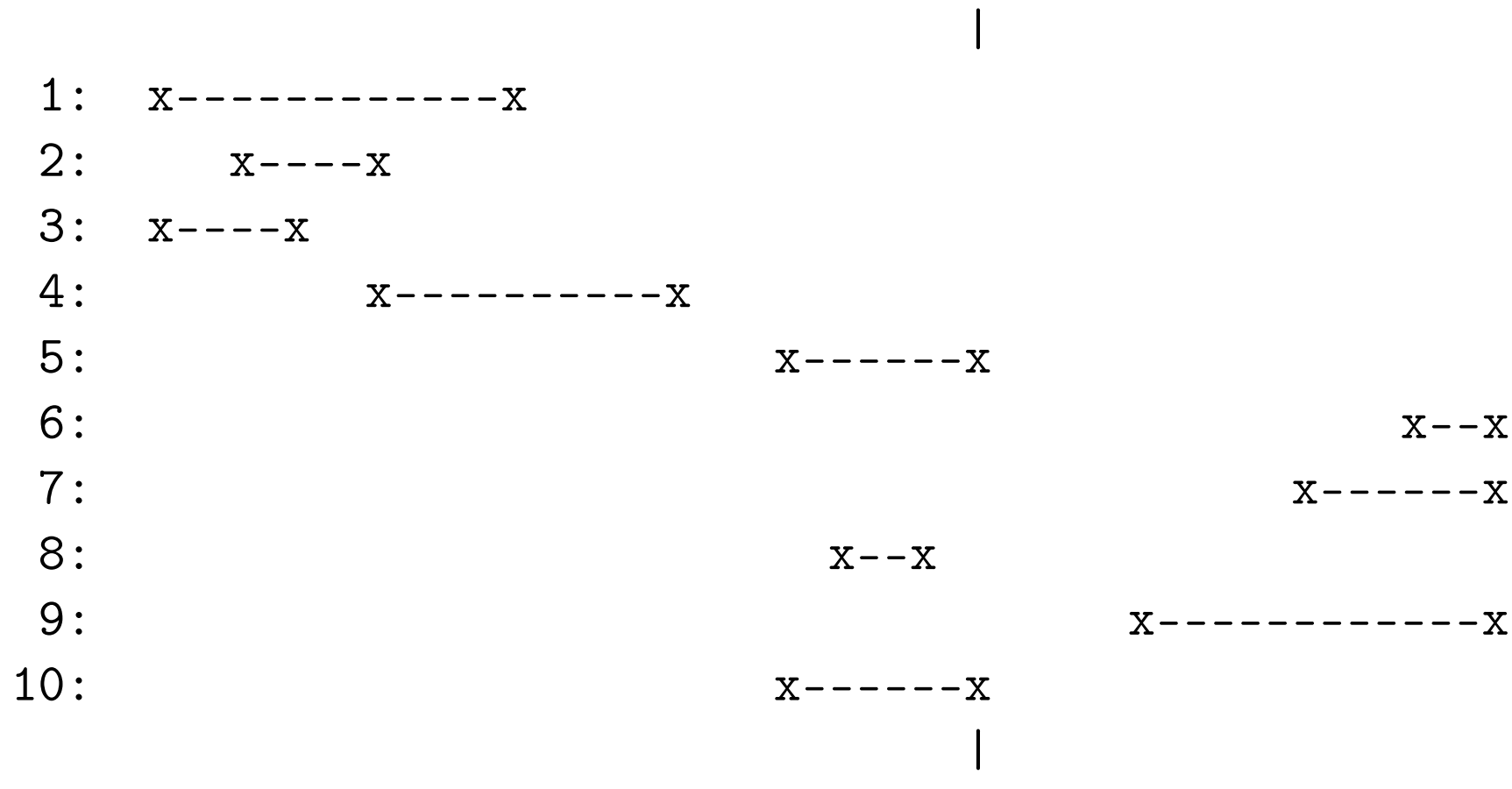
[5]

r = 20



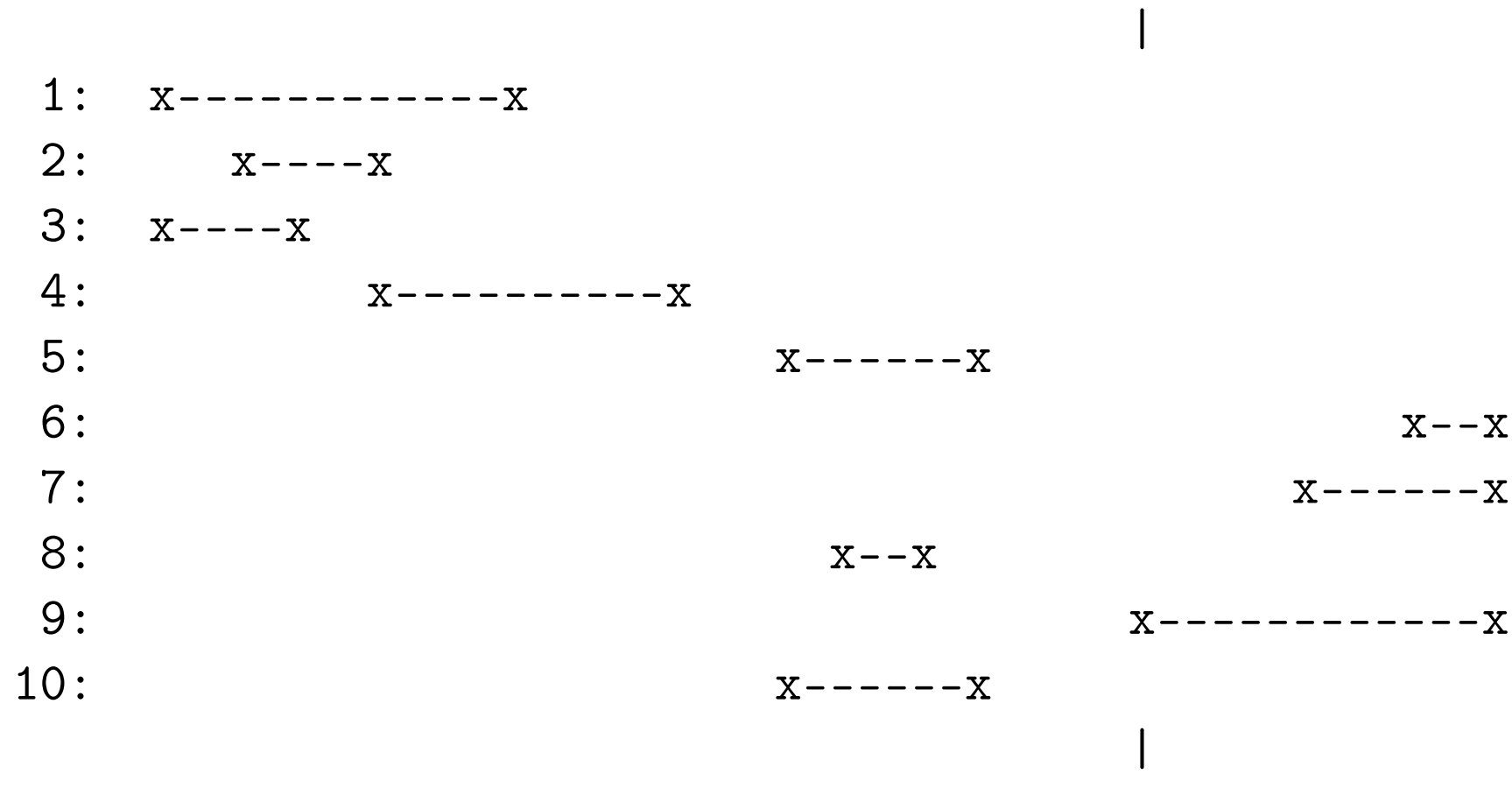
[]

r = 20



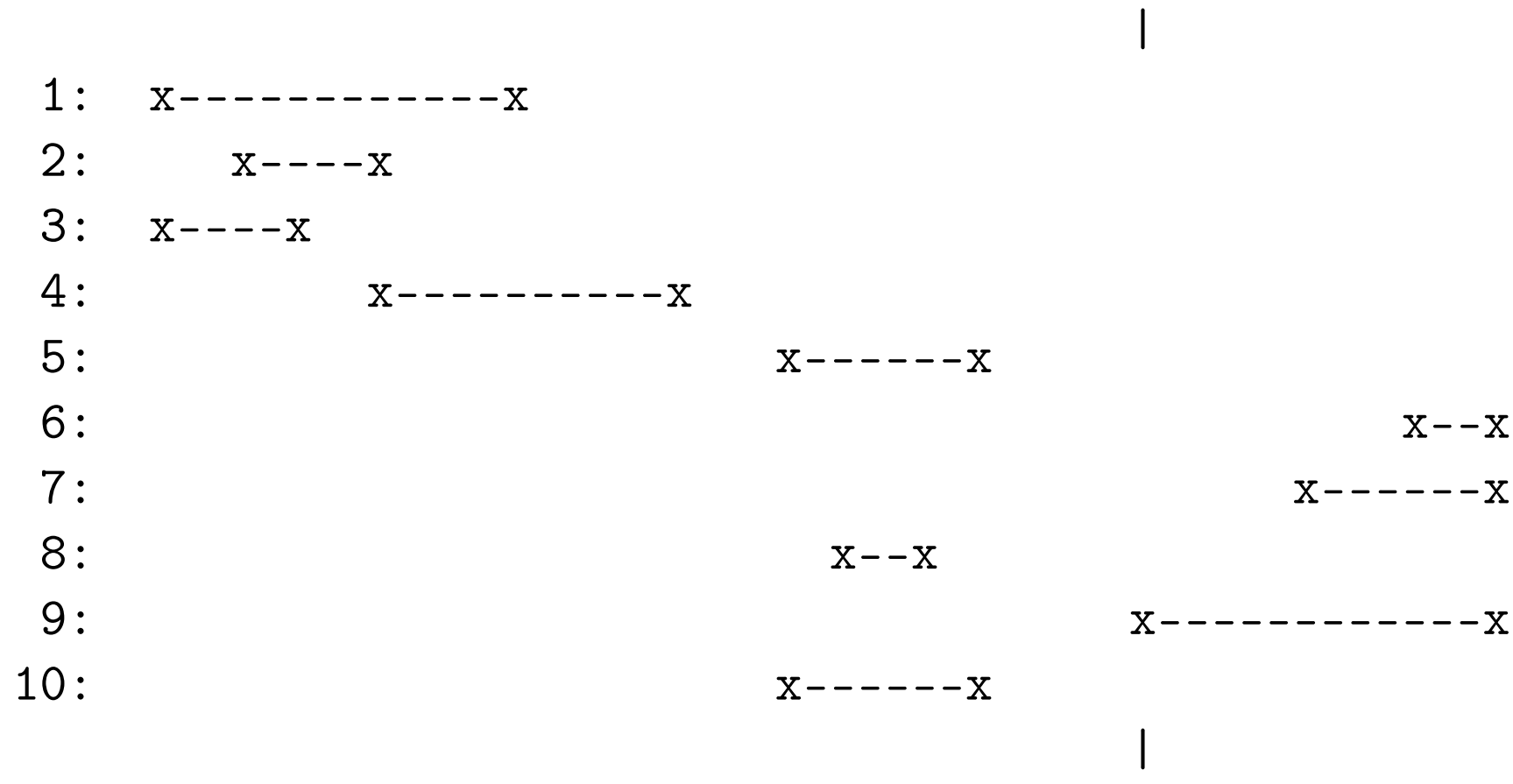
[]

r = 28



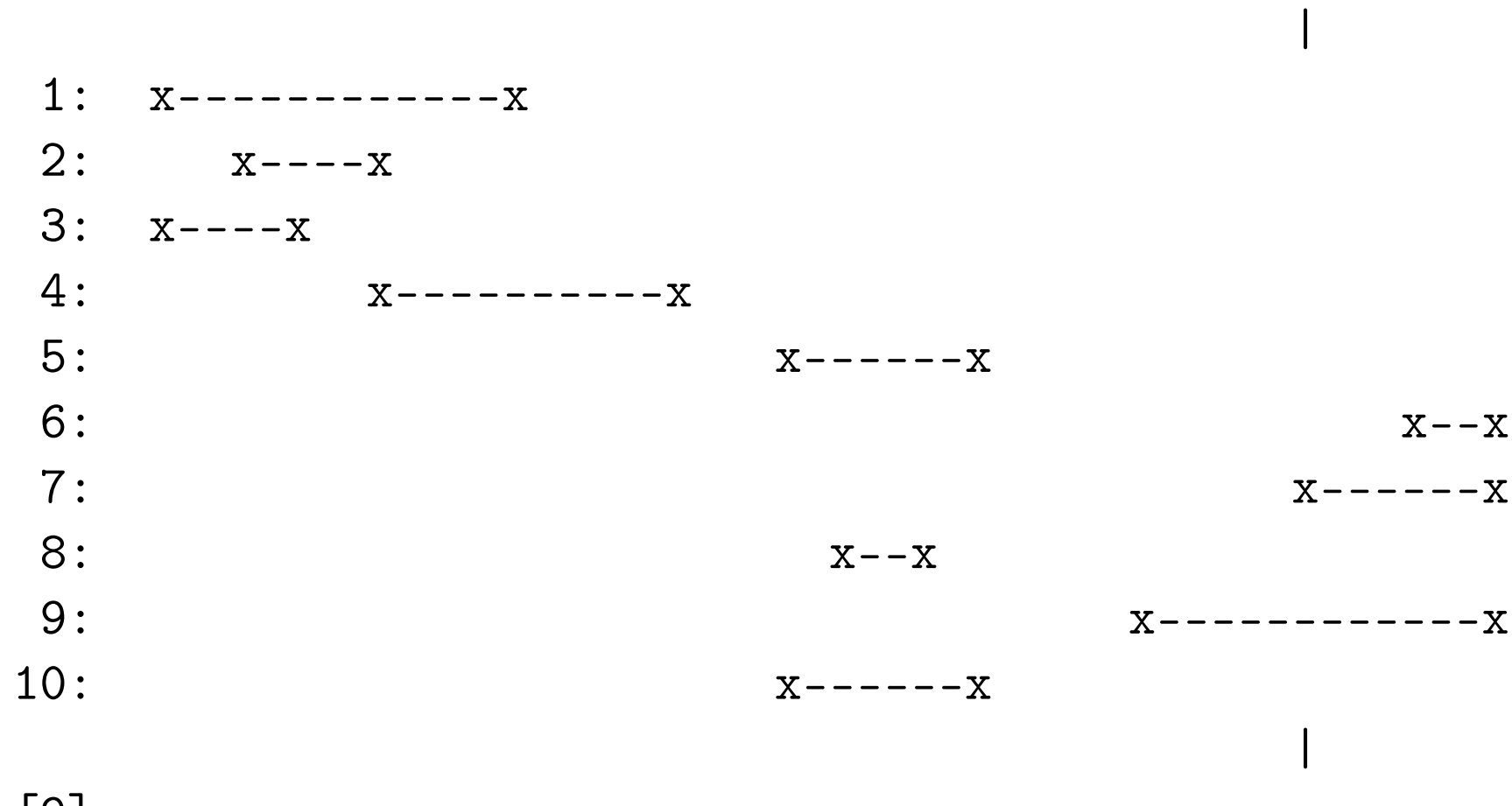
[]

r = 28



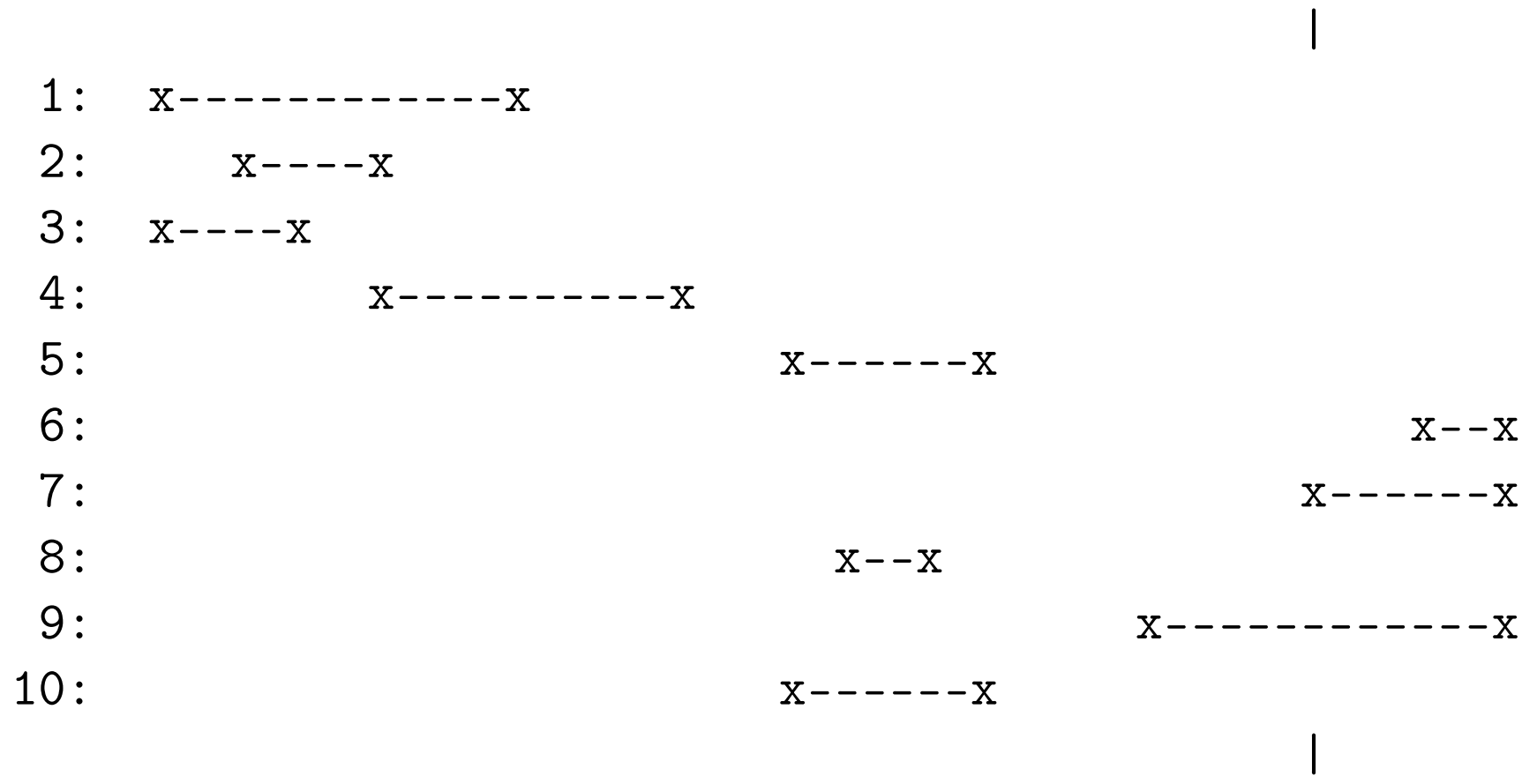
[9]

r = 28

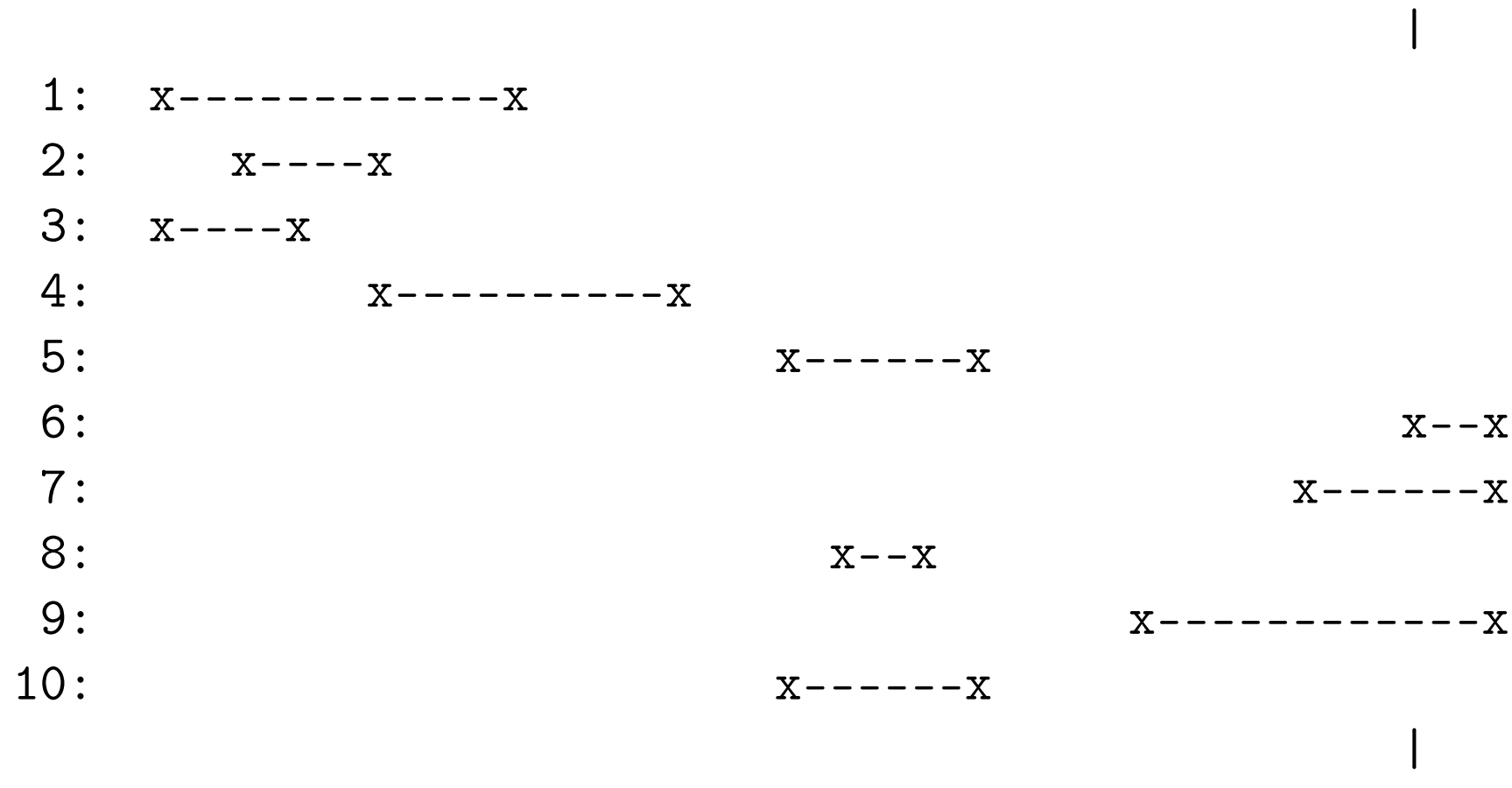


[9]

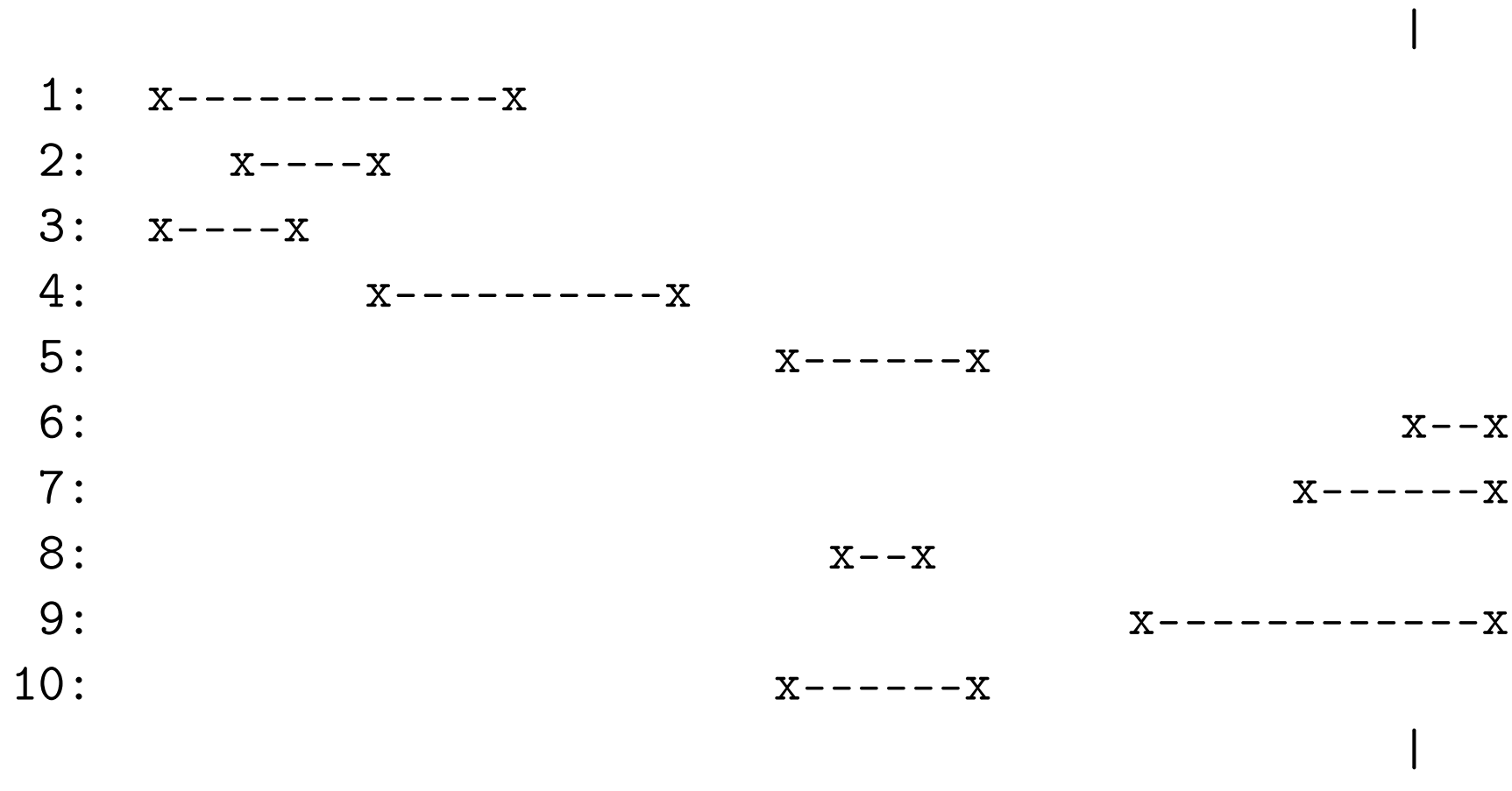
r = 28



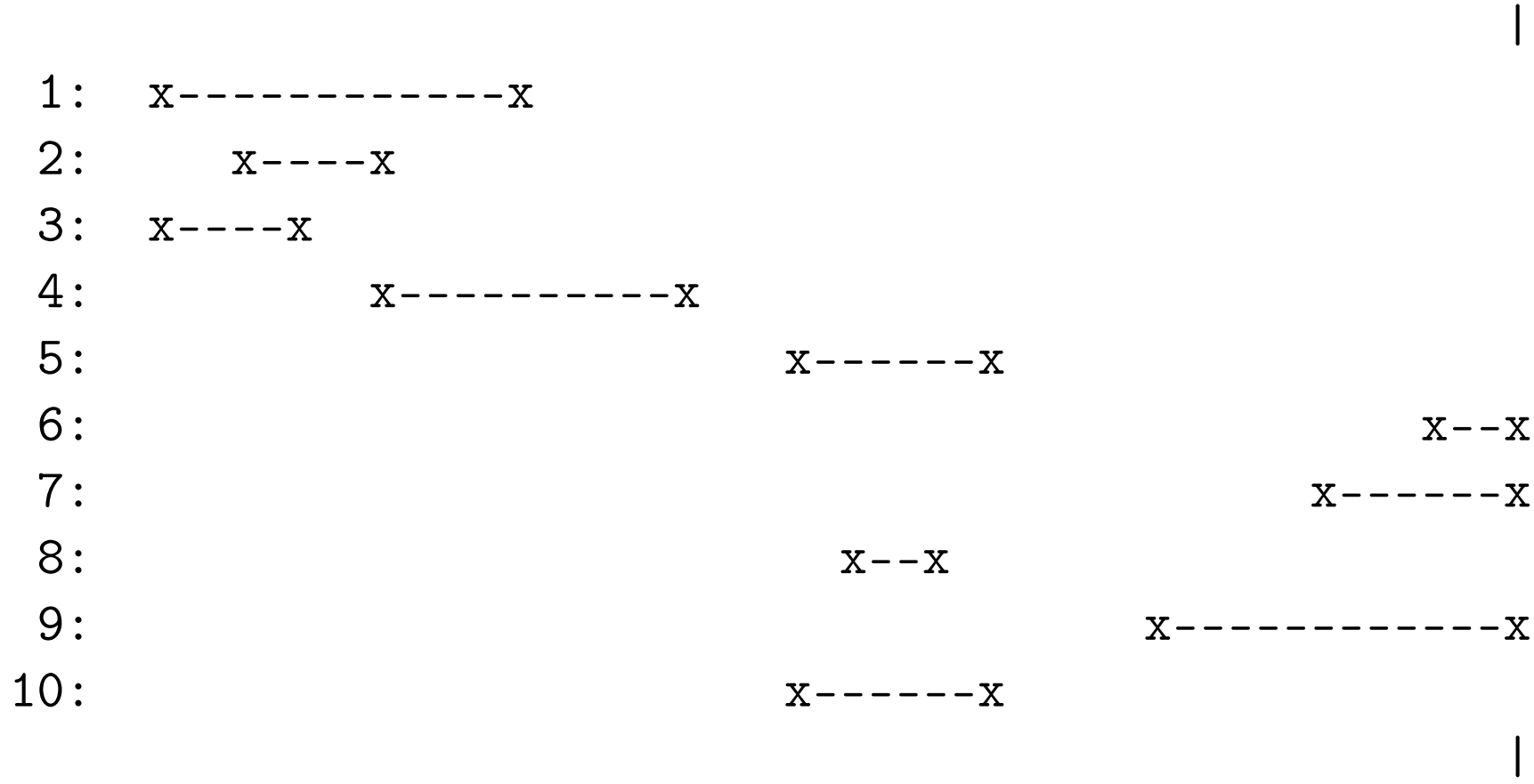
[7, 9]
r = 28



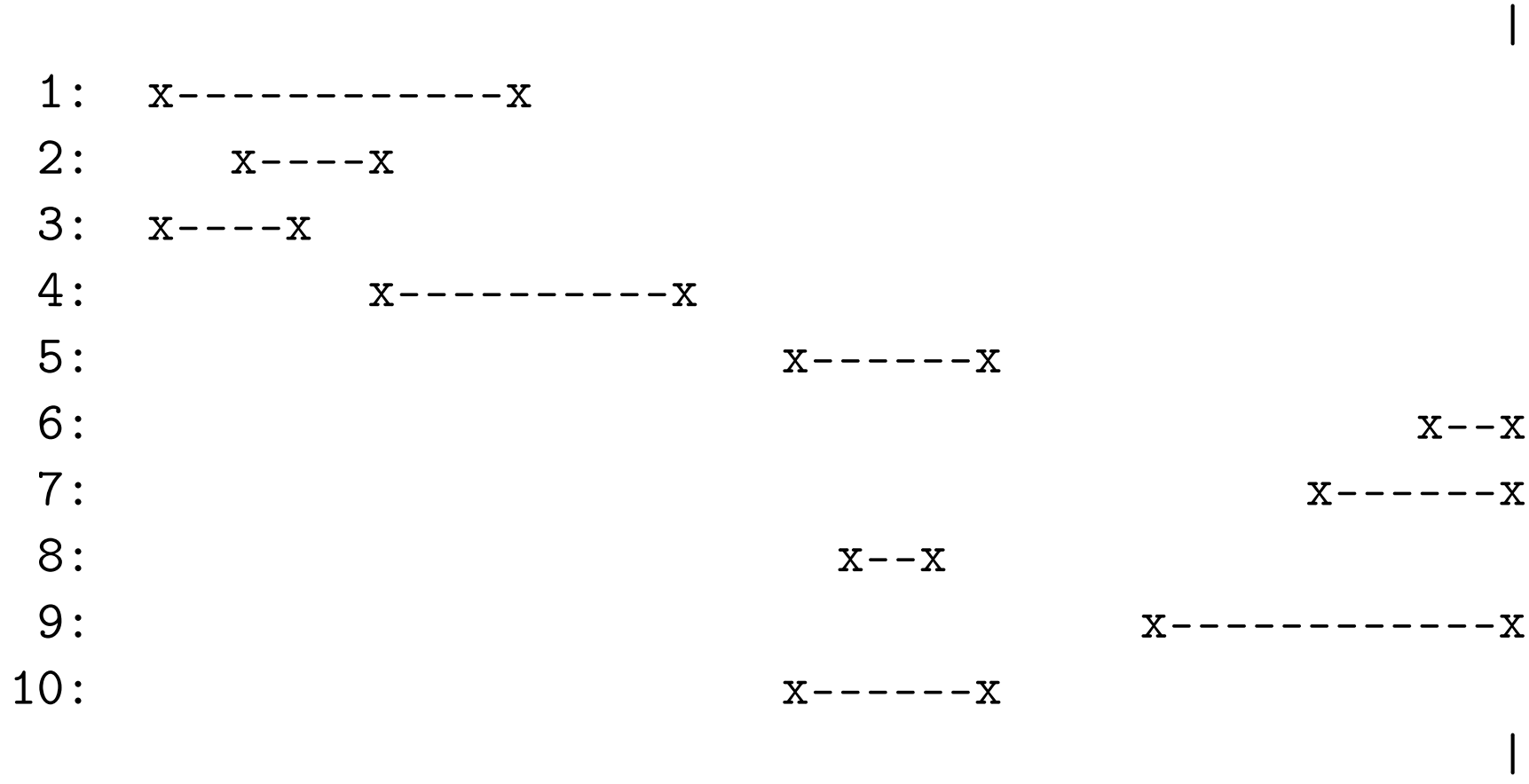
[7, 9]
r = 28



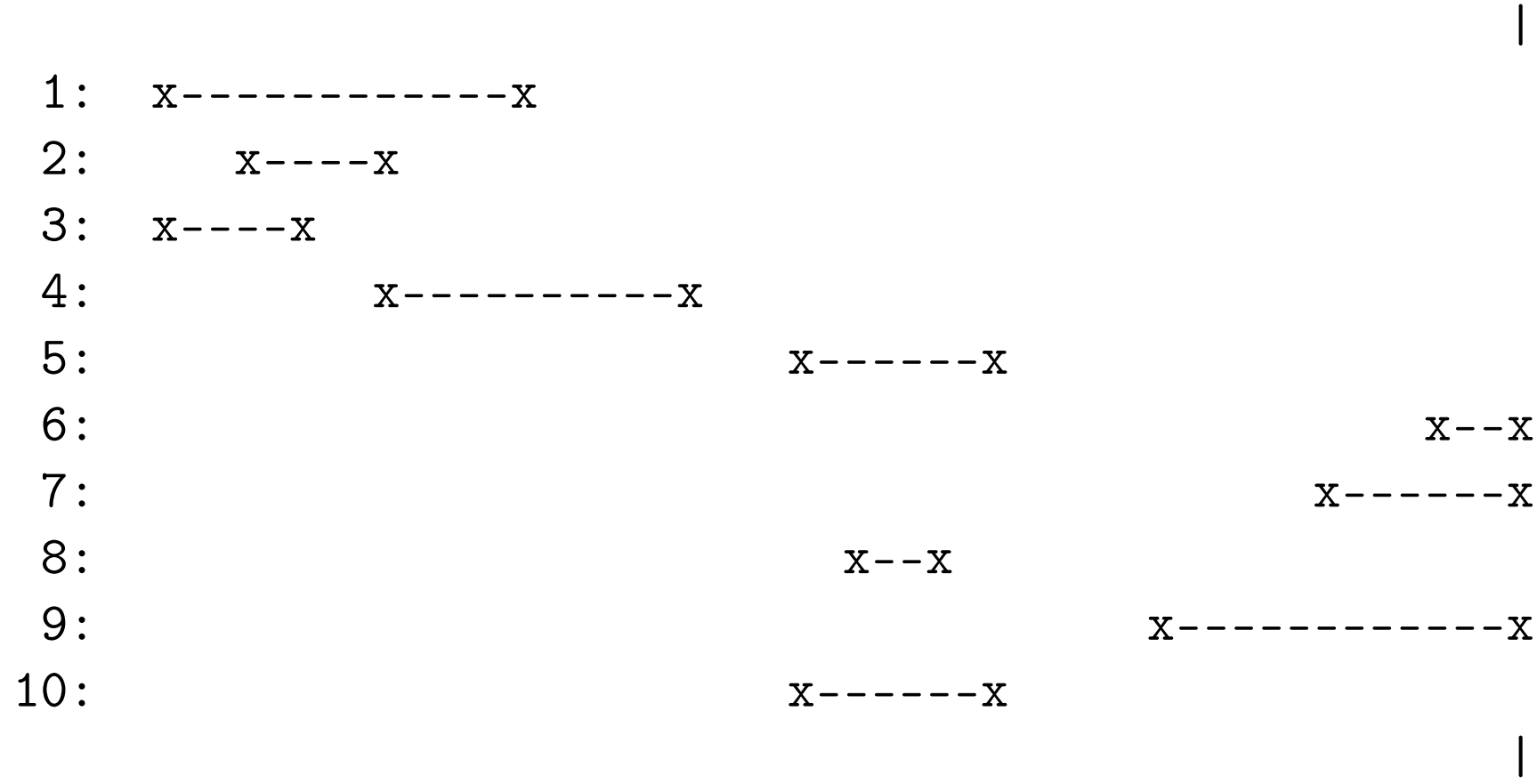
[6, 7, 9]
r = 28



[6, 7, 9]
r = 28

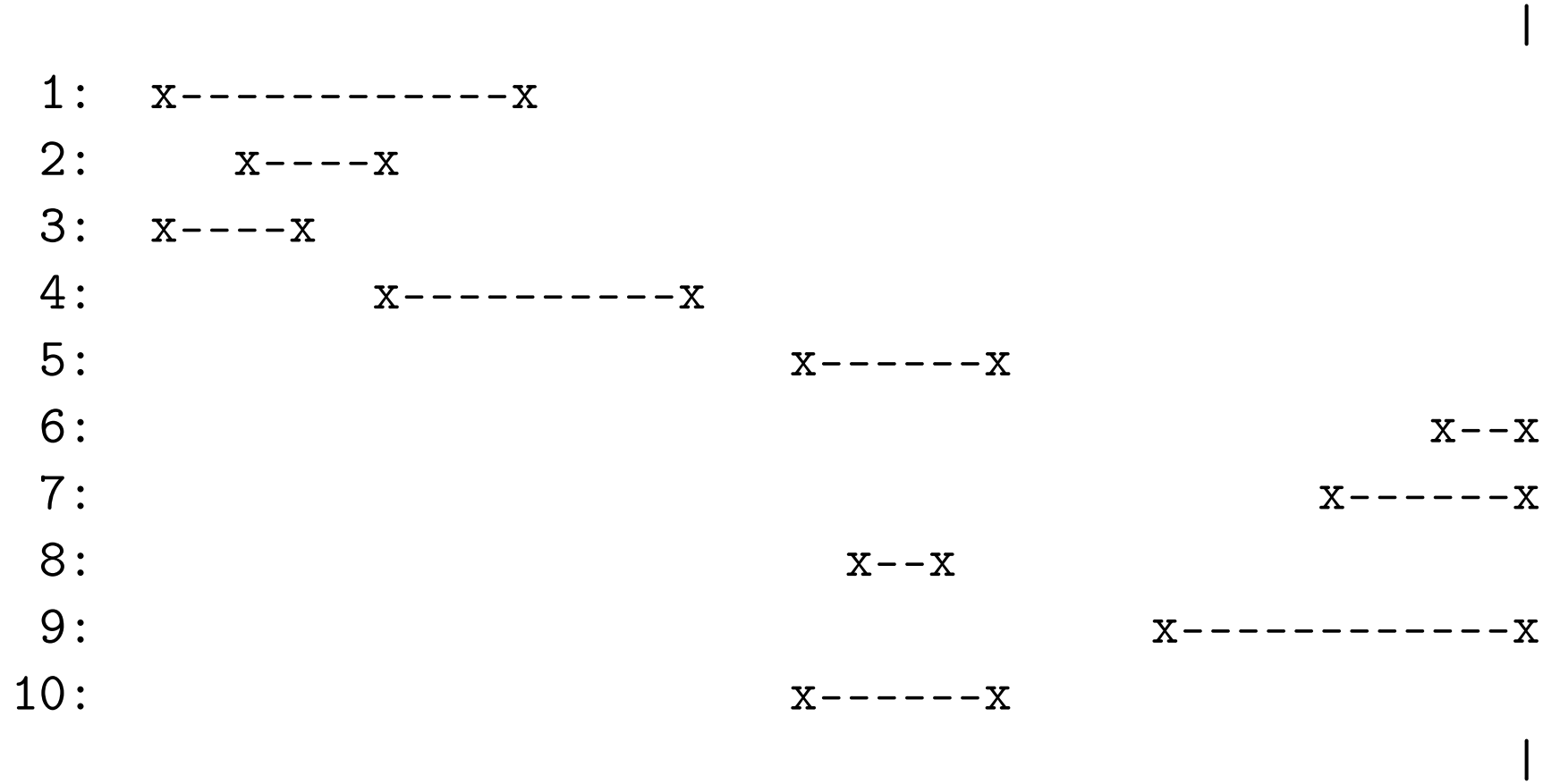


[7, 9]
r = 28



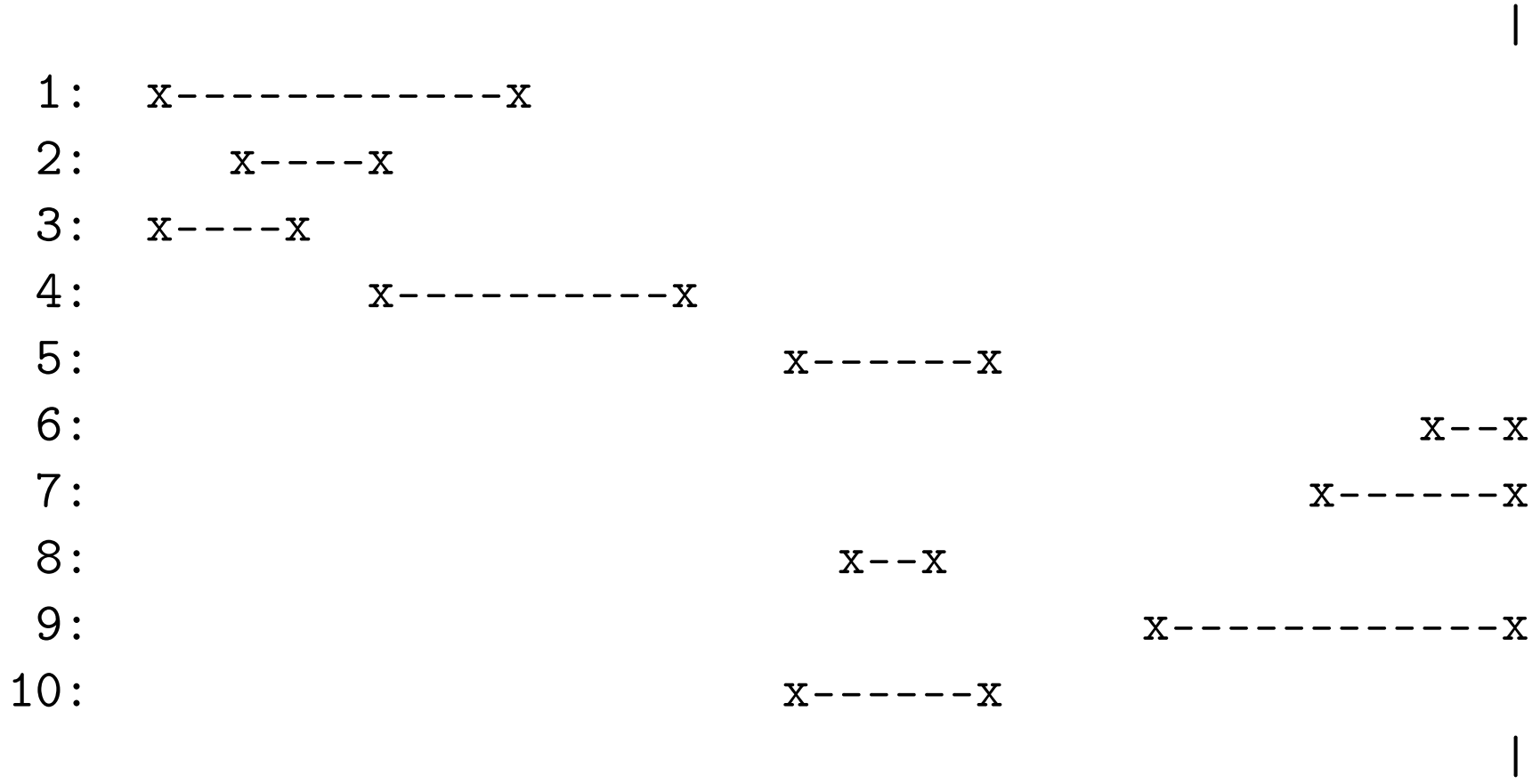
[9]

r = 28



[]

r = 28




```
[]
r = 42
```

```
3  typedef struct { int x, y; } ii;
4  int cmp(const void* p1, const void* p2) { return ((ii*)p1)->x - ((ii*)p2)->x; }
5
6  int main()
7  {
8      int n, r, i, j, k;
9      scanf("%d", &n);
10     ii a[2*n];
11     int b[n];
12     for (i = 0; i < n; i++)
13     {
14         scanf("%d%d", &(a[2*i].x), &(a[2*i + 1].x));
15         a[2*i].y = i; a[2*i + 1].y = i; b[i] = 0;
16     }
17     qsort(a, 2*n, sizeof(a[0]), cmp);
18     i = 0, r = 0;
19     while (i < 2*n)
20     {
21         k = 1, j = i + 1, b[a[i].y] = 1;
22         while (k > 0)
23         {
24             if (b[a[j].y] == 1) k--;
25             else b[a[j].y] = 1, k++;
26             j++;
27         }
28         r = r + a[j - 1].x - a[i].x; i = j;
29     }
30     printf("%d\n", r);
31     return 0;
32 }
```

- Við byrjum á að raða í $\mathcal{O}(n \log n)$ tíma.
- Síðan ítrum við í gegnum alla endapunktana sem tekur $\mathcal{O}(\, n \,)$ tíma.
- Svo lausnin hefur tímaflækjuna $\mathcal{O}(n \log n)$.