Samansóp

Bergur Snorrason

13. apríl 2021

Lokakeppnin

- Í næstu viku verður lokakeppnin, og þar með síðasti tíminn í þessu námskeiði.
- Mér líst best á að hafa keppnina í miðvikudagstímanum okkur.
- Við höfum þá stoðtíma í mánudagstímanum.
- Í keppninni verða fimm dæmi.
- Skil fást fyrir að leysa eitt dæmi.
- Ef þið leysið þrjú þeirra fáið þið aukaskil.

Strengjaleit

- Gefum okkur langan streng `s` og styttri streng `p`.
- Hvernig getum við fundið alla hlutstrengi `s` sem eru jafnir `p`.
- Fyrsta sem manni dettur í hug er að bera `p` saman við alla hlutstrengi `s` af sömu lengd og `p`.

```
1  void naive(char* s, int n, char* p, int m)
2  {
3      int i, j;
4      rep(i, n - m + 1)
5      {
6          rep(j, m) if (s[i + j] != p[j]) break;
7          if (j >= m) printf("%d ", i - j);
8      }
9      printf("\n");
10 }
```

- Gerum ráð fyrir að `s` sé af lengd n og `p` sé af lengd m.
- Fjöldi hlutstrengja í `s` að lengd m er $n - m + 1$.
- Strengja samanburðurinn tekur línulegan tíma.
- Svo tímaflækja leitarinnar er $\mathcal{O}(nm - m^2)$.
- Ef $m = n/2$ þá er $nm - m^2 = n^2/2 - n^2/4 = n^2/4$ tímaflækjan er í raun $\mathcal{O}(n^2)$.
- Dæmi um leiðinlega strengi væri `s = ‘‘aaaaaaaaaaaaaaaa’’` og `p = ‘‘aaaaaaab’’`.

- Þó þessi aðferð sé ekki góð þá er hún stundum nógu góð.
- Það er þó óþarfi að útfæra hana því hún fylgir með flestum forritunarmálum, til dæmis:
  - Í string.h í `C` er `strstr(..)`.
  - Í string í `C++` er `find(..)`.
  - Í String í `Java` er `indexOf(..)`.
- Munið bara að ef $n > 10^4$ er þetta yfirleitt of hægt.

Reiknirit Knuth, Morrisar og Pratts (KMP) strengjaleit (1970)

- Er einhver leið til að bæta strengjaleitina úr fyrri glærum?
- Skoðum betur sértilfellið `p` = ‘‘`aaaabbbb`’’.
- Ef strengja samanburðurinn misheppnast í `p[3]` þá myndi einfalda strengjaleitin okkar hliðra `p` um einn og reyna aftur.
- En við vitum að fyrstu tveir stafnirnir í næsta hlutstreng stemma, svo við getum byrjað í `p[2]`.
- Reiknirit Knuths, Morrisar og Pratts notar sér þessa hugmynd til að framkvæma strengjaleit.
- Reikniritið byrjar á að forreiknað hversu mikið maður veit eftir misheppnaðan samanburð.

```
12 void bkmp(char* p, int* b, int m)
13 { // Forreikningar fyrir kmp(...).
14     int i = 0, j = -1;
15     b[0] = -1;
16     while (i < m)
17     {
18         while (j >= 0 && p[i] != p[j]) j = b[j];
19         b[++i] = ++j;
20     }
21 }
```

- Svo þurfum við einfaldlega að labba í gegnum s og hliðra eins og á við.

```
23 void kmp(char* s, int n, char* p, int m, int* b)
24 { // Utfaersla a reikniriti Knuths, Morrisar og Pratts.
25     int i = 0, j = 0;
26     while (i < n)
27     {
28         while (j >= 0 && s[i] != p[j]) j = b[j];
29         i++, j++;
30         if (j == m) printf("%d\n", i - j), j = b[j];
31     }
32 }
```

- Takið eftir að hver ítrun innri lykkjanna svarar til einnar ítrunar ytri lykkjanna.
- Svo innri lykkjan keyrir, í heildina, ekki oftar en ytri lykkjan.
- Því fæst að tímaflækja forreikninganna $\mathcal{O}(\, m\,)$.
- Eins er tímaflækja strengjaleitarinnar $\mathcal{O}(\, n\,)$.
- Saman er því tímaflækjan $\mathcal{O}(n + m)$.

Reiknirit Ahos og Corasicks (1975)

- Til er önnur aðferð, svipuð og `KMP`, sem finnur staðsetningar margra orða í einu í streng.
- Hún er kennd við Aho og Corasick.
- Ég fer ekki í hana hér en hún byggir á því að gera stöðuvél.
- Reikniritið keyrir í línulegum tíma í lengd allra strengjanna, ásamt fjölda heppnaðra samanburða.

Hlaupabil

- Aðferð hlaupabila (e. sliding window) er stundum hægt að nota til að taka dæmi sem hafa augljósa $\mathcal{O}(n^2)$ 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 lengsta bils í rununni $(a_n)_{n\in\mathbb{N}}$ sem inniheldur 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 lengsta bili í $(a_n)_{n\in\mathbb{N}}$ sem hefur mesta 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]
|   |
```

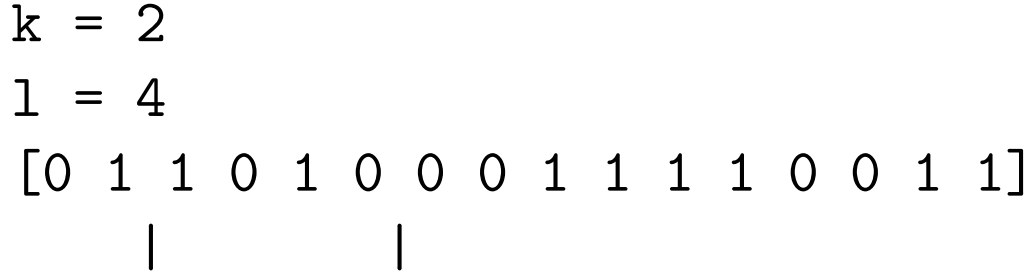
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|     |
```

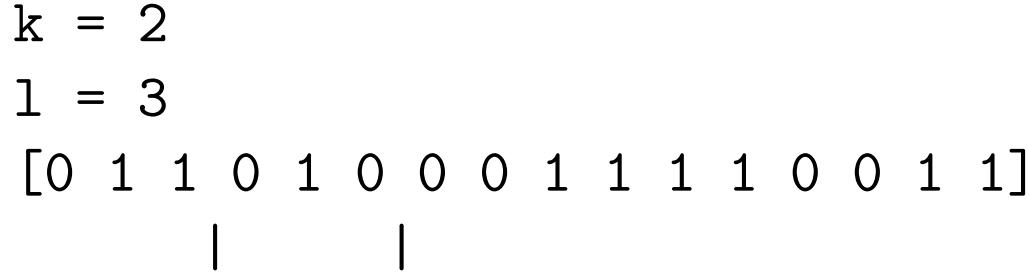
```
k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|       |
```

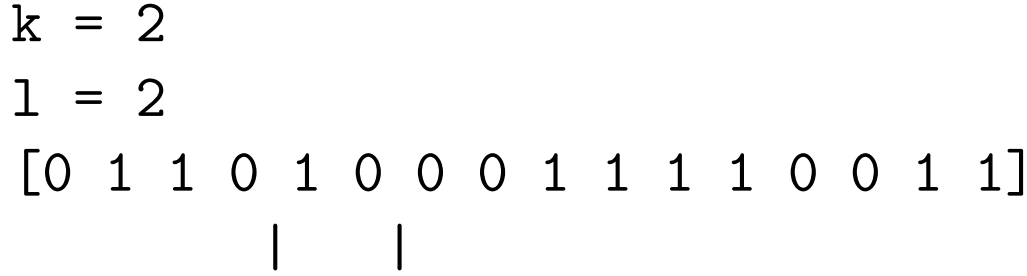
```
k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
|         |
```

```
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]
| |

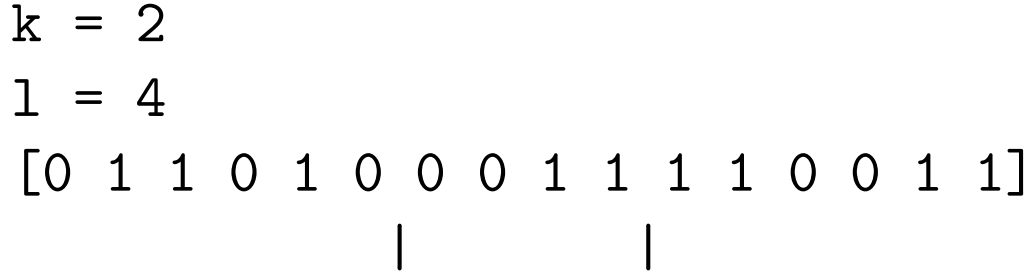
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
         |       |
```

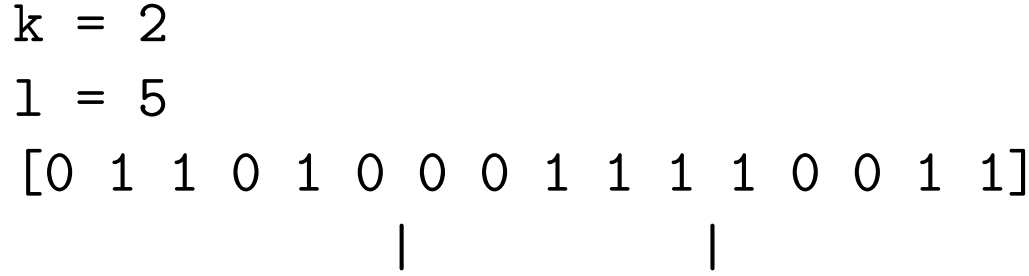
```
k = 2
l = 2
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
           |   |
```

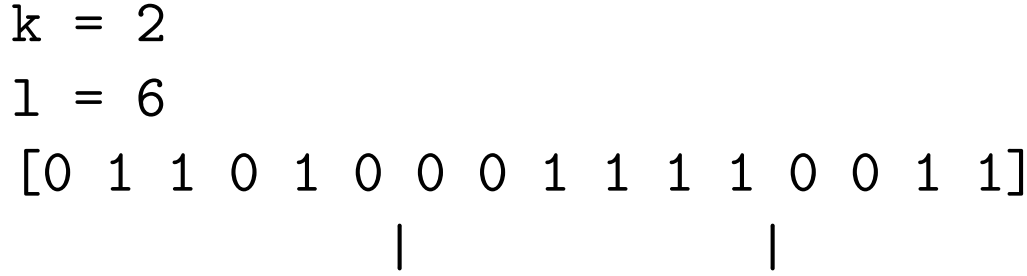
```
k = 2
l = 1
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
             | |
```

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

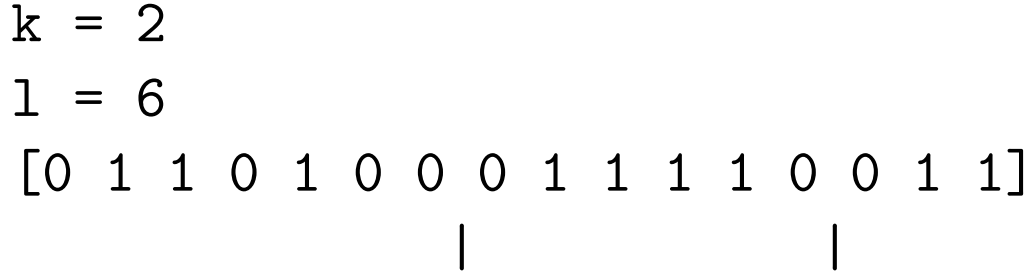
```
k = 2
l = 3
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
             |       |
```

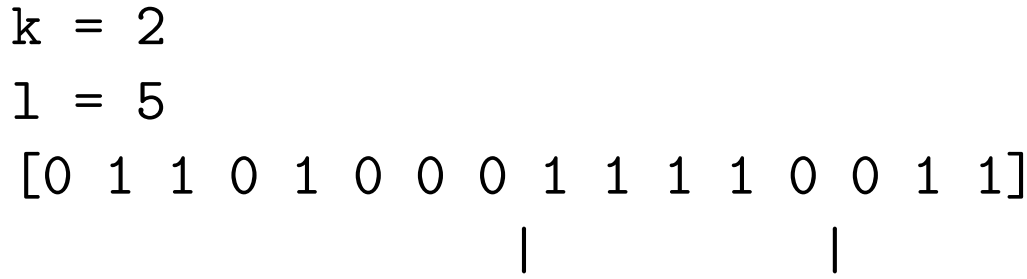

k = 2
l = 4
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

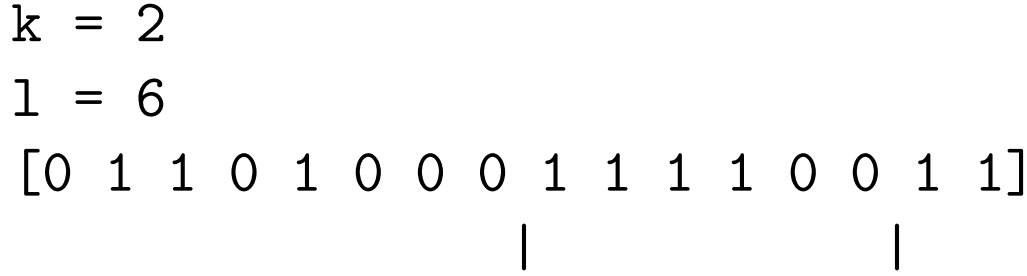

k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

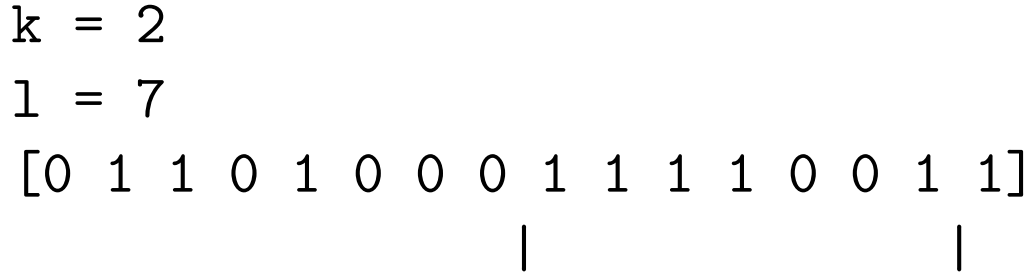

k = 2
l = 6
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]

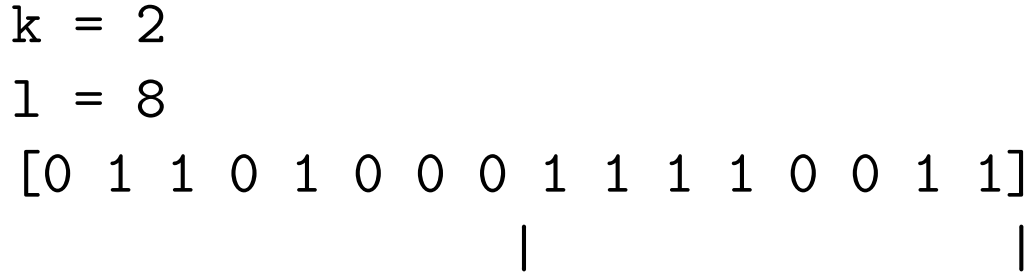
```
k = 2
l = 5
[0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1]
                  |         |
```


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]

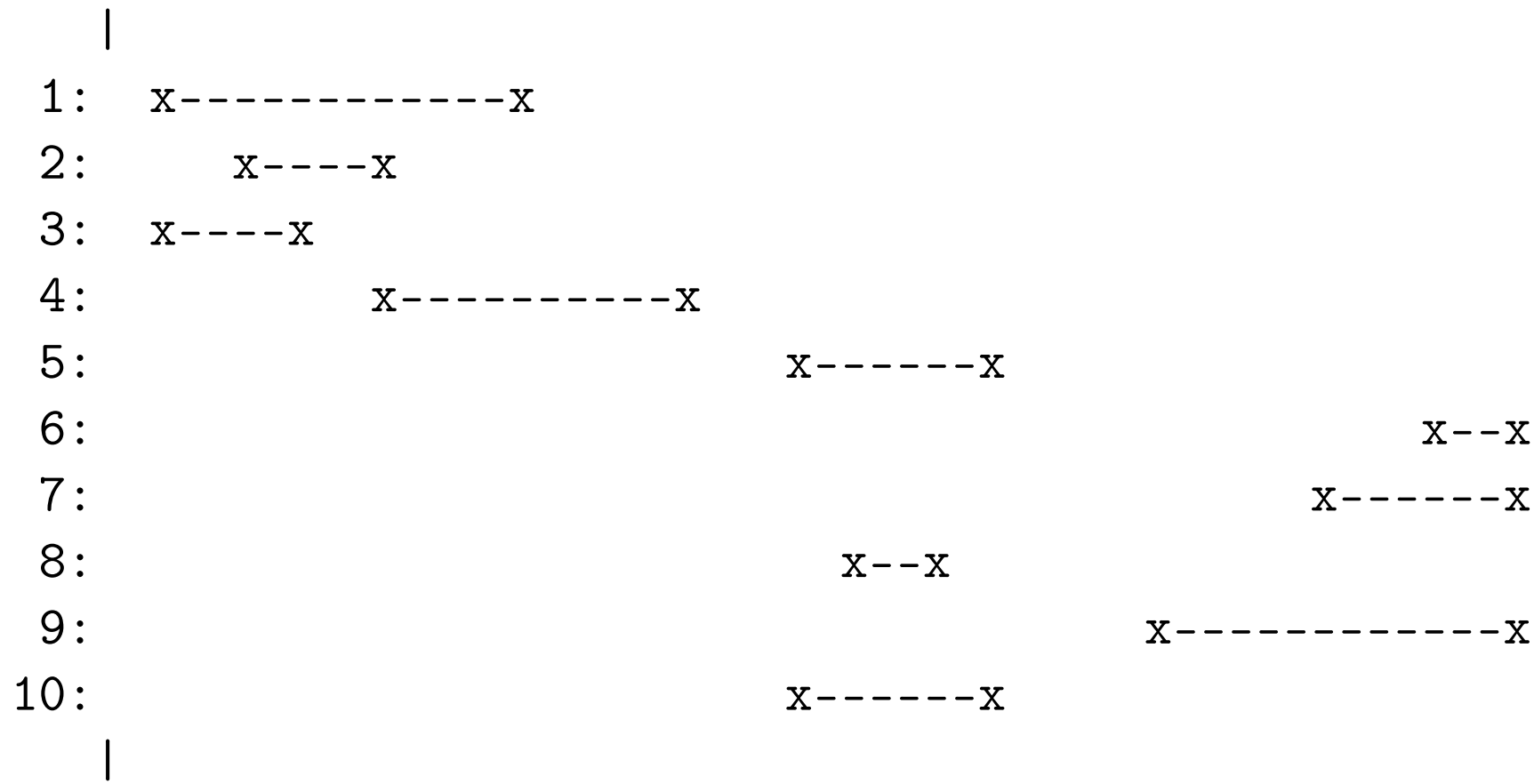
```c
#include <stdio.h>
#define rep(E, F) for (E = 0; E < (F); E++)

int main()
{
    int n, k, i;
    scanf("%d%d", &n, &k);
    int a[n];
    rep(i, n) scanf("%d", &(a[i]));
    int b = 0, z = 0, mx = 0;
    rep(i, n)
    {
        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;
}
```

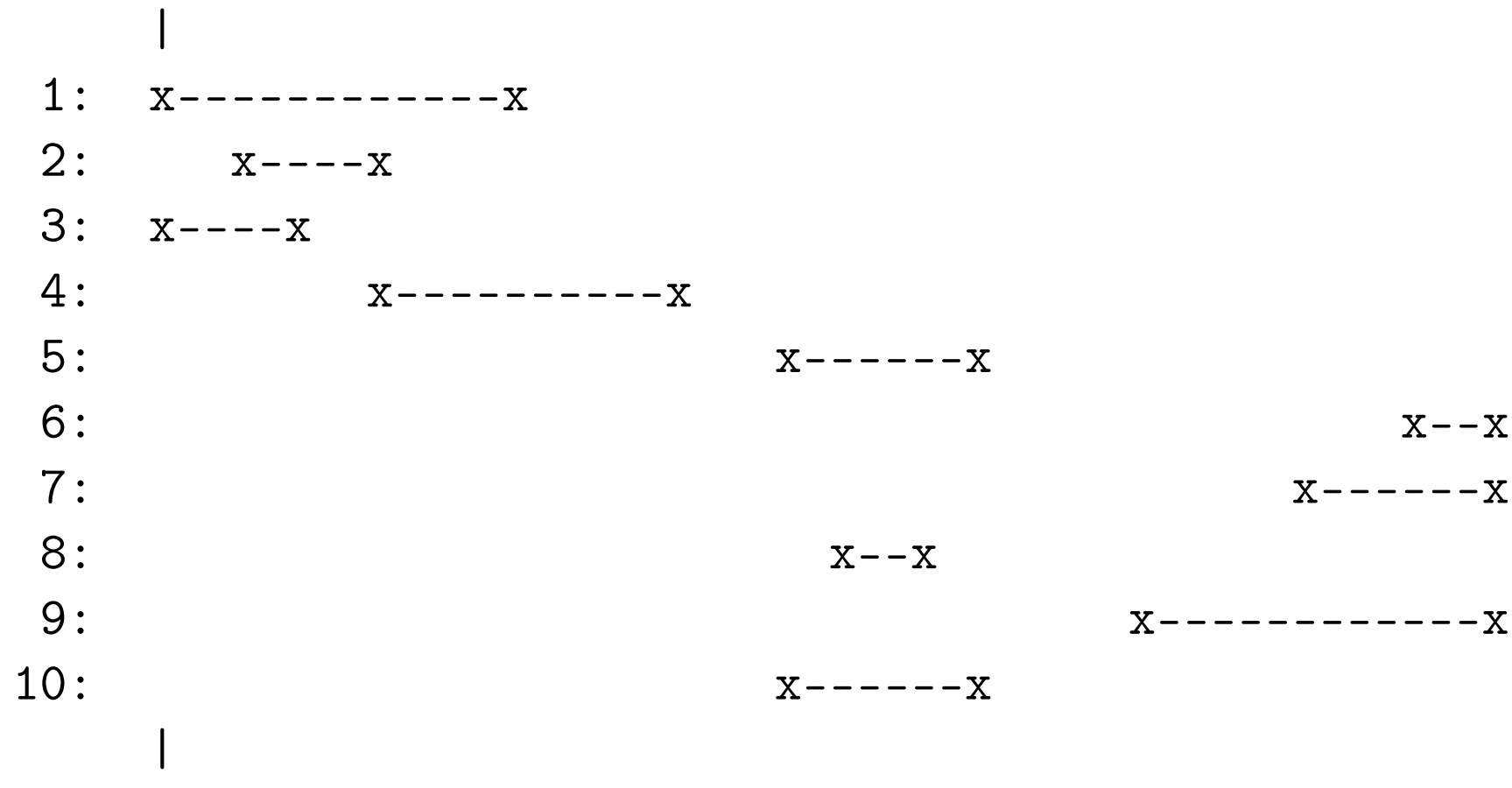
- Hver tala í rununni er sett einu sinni í hlaupabilið og mögulega fjarlægð úr því.
- Svo tímaflækjan er $\mathcal{O}(\, n \,)$.

- Þetta dæmi er nú 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.

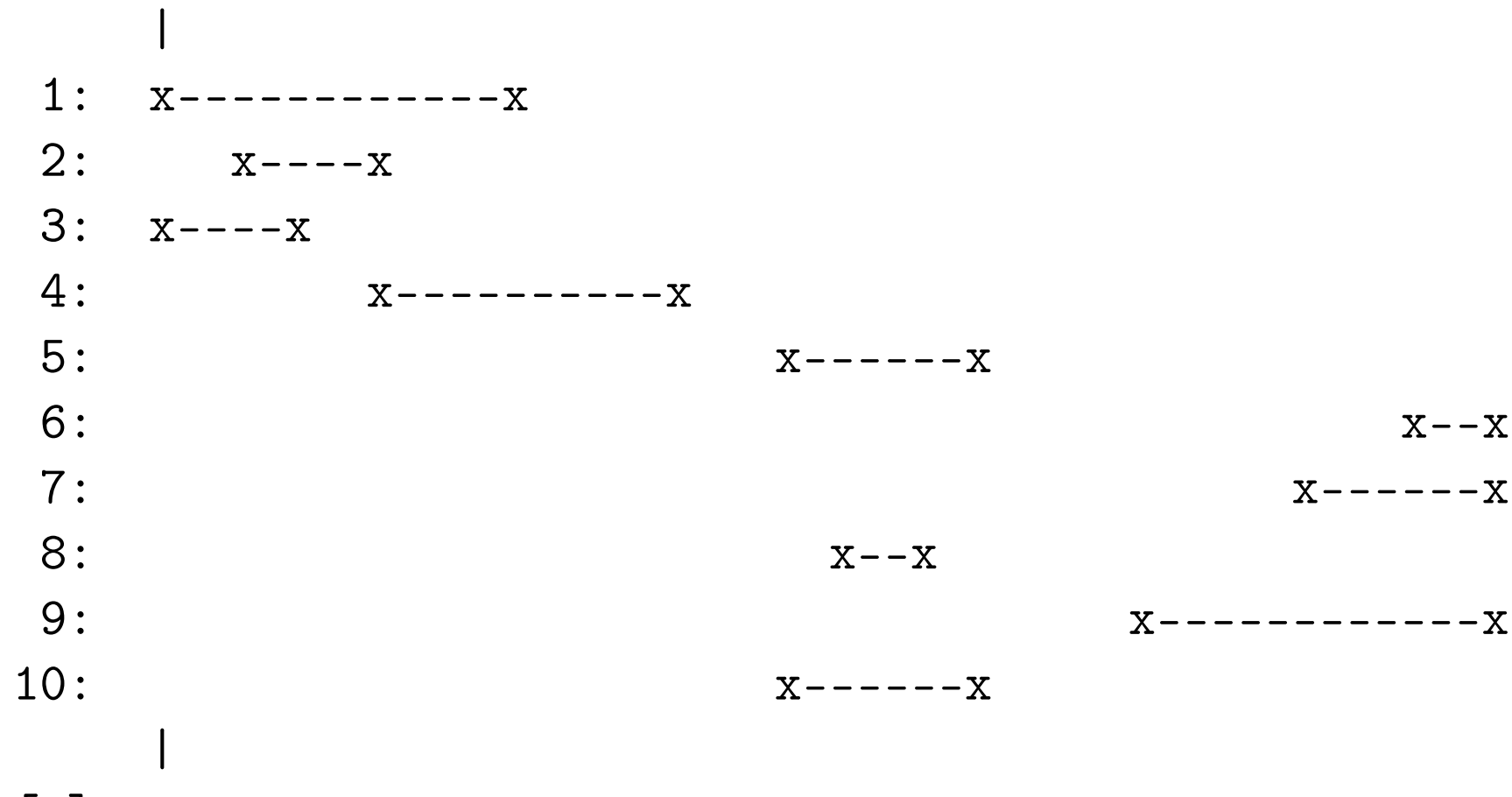
- Geymum í lista tvenndir þar sem fyrra stakið er endapunktur bils og seinna stakið segir hvaða bili punkturinn tilheyrir.
- Röðum þessum punktum svo í vaxandi röð.
- Við löbbum í gegnum þennan raðaða lista og höldum utan um hlaupabil þannig að við bætum við bili í hlaupabilið þegar við rekumst á vinstri 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 okkur.
- Sammengi þeirra bila sem við höfum farið í gegnum þá síðan hlaupabilið var síðast tómt er nú sundurlægt öllum öðrum bilum sem okkur var gefið í byrjun.
- Við skilum því summu lengda þessara sammengja.

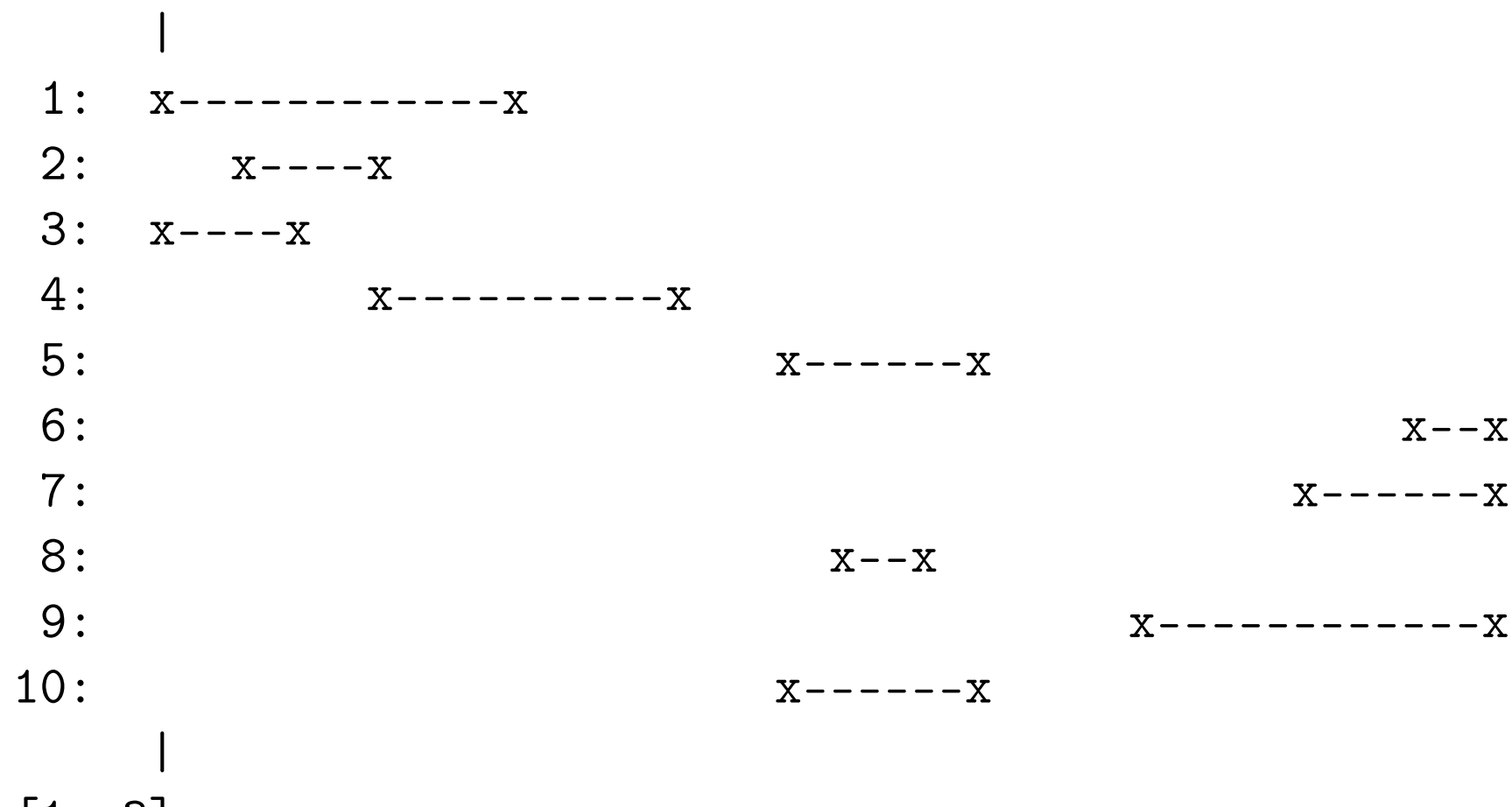

1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[]
r = 0

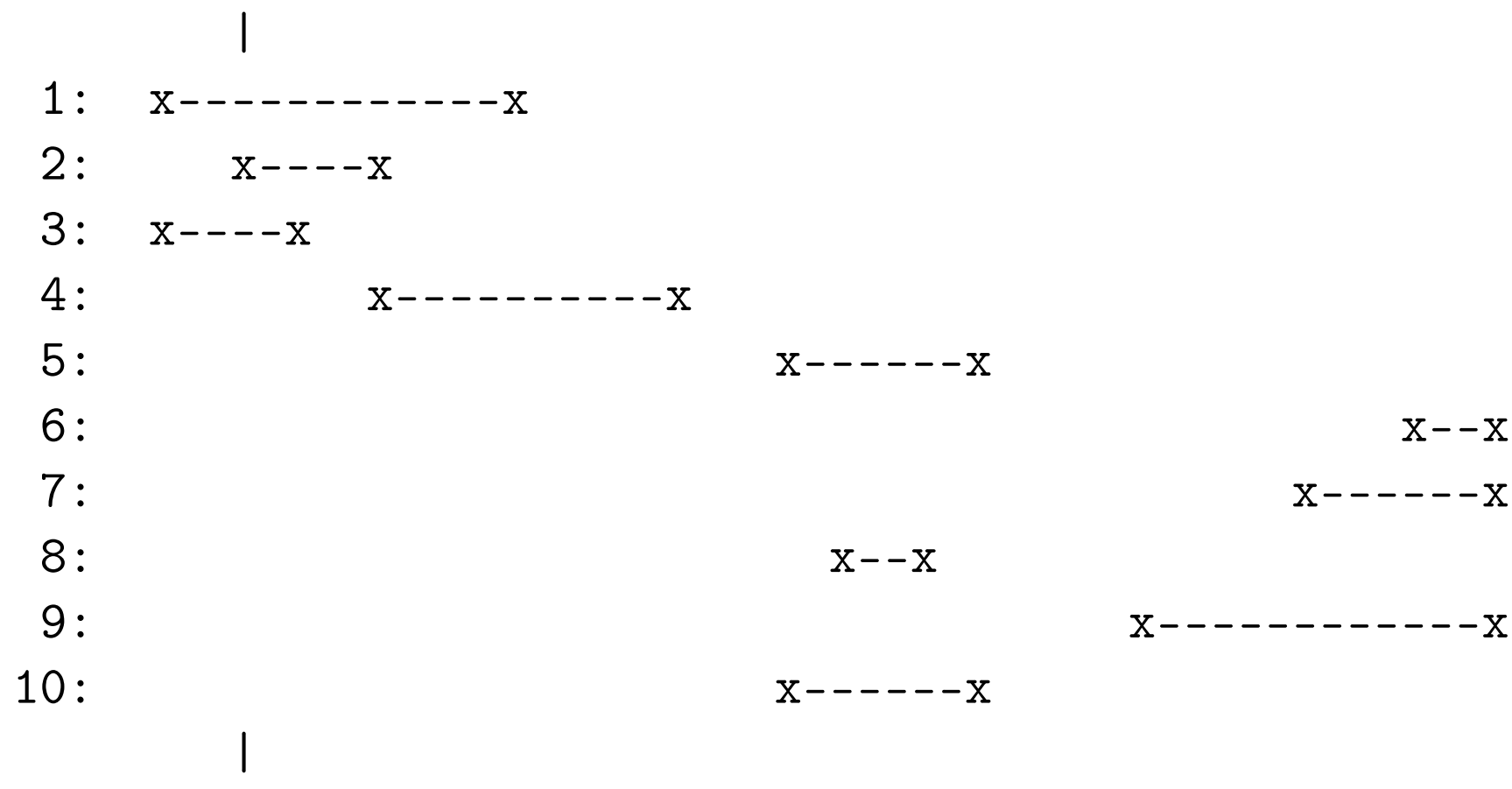


[]

r = 0

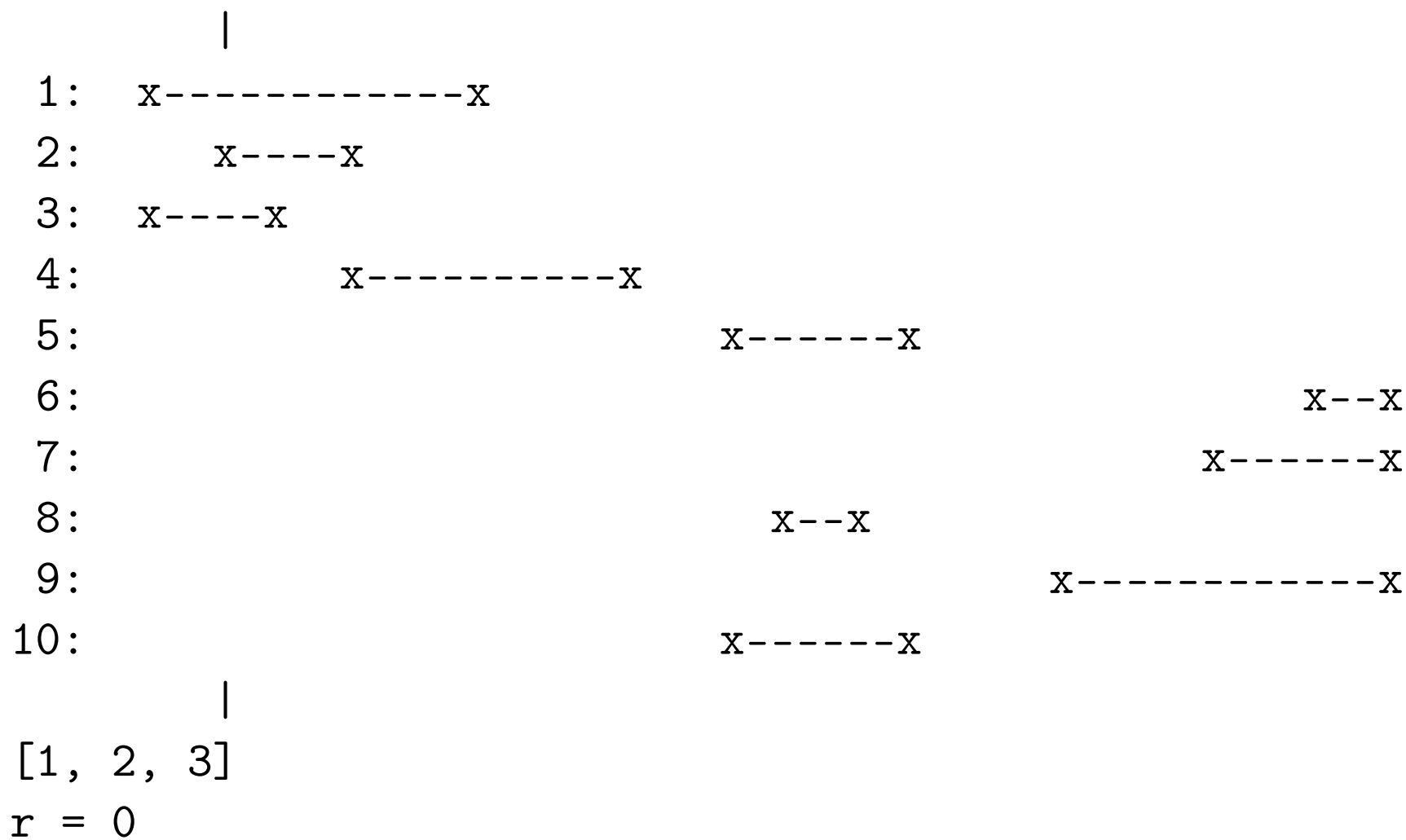

1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[1]
r = 0

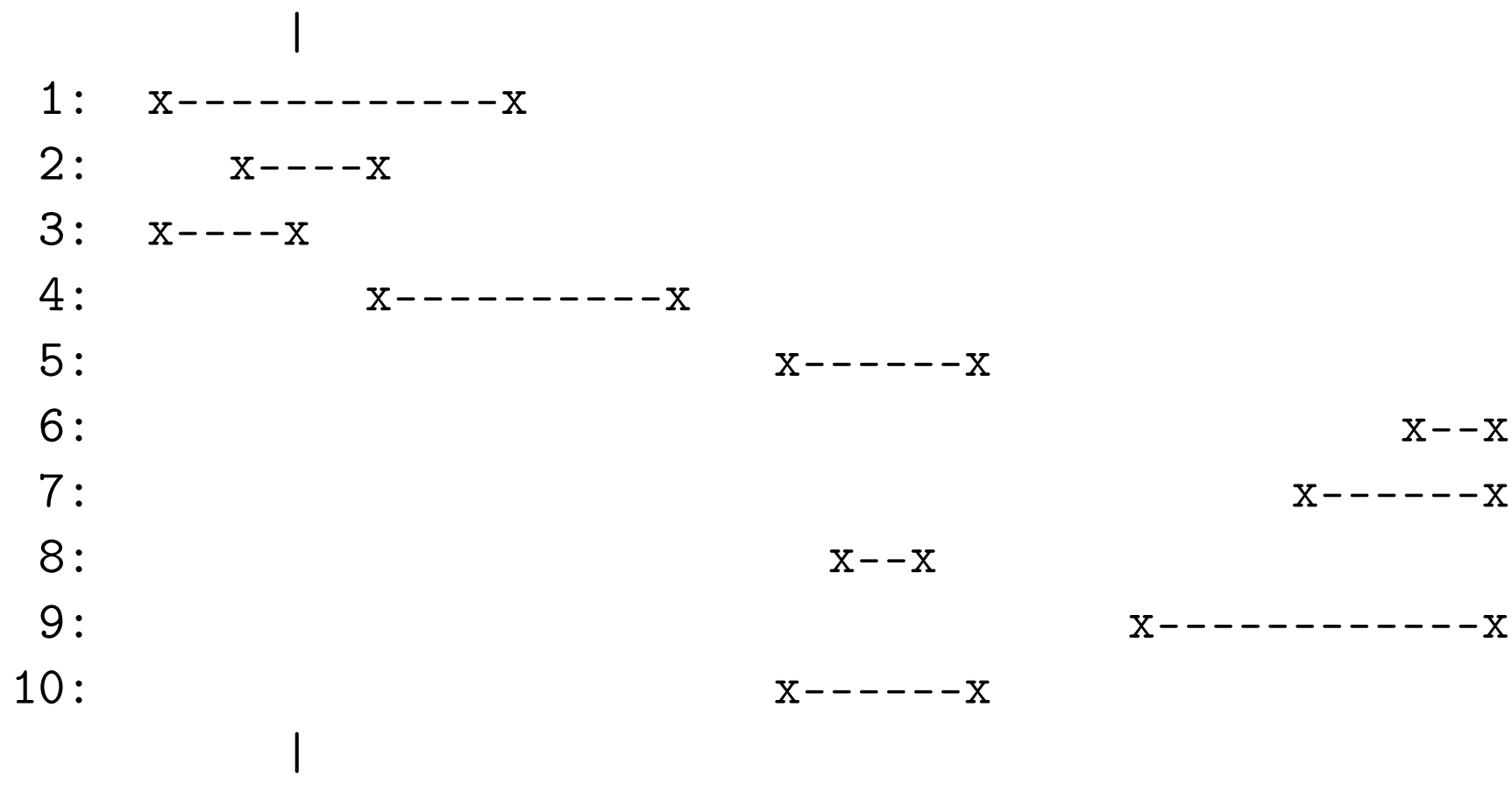

1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
[1, 3]
r = 0



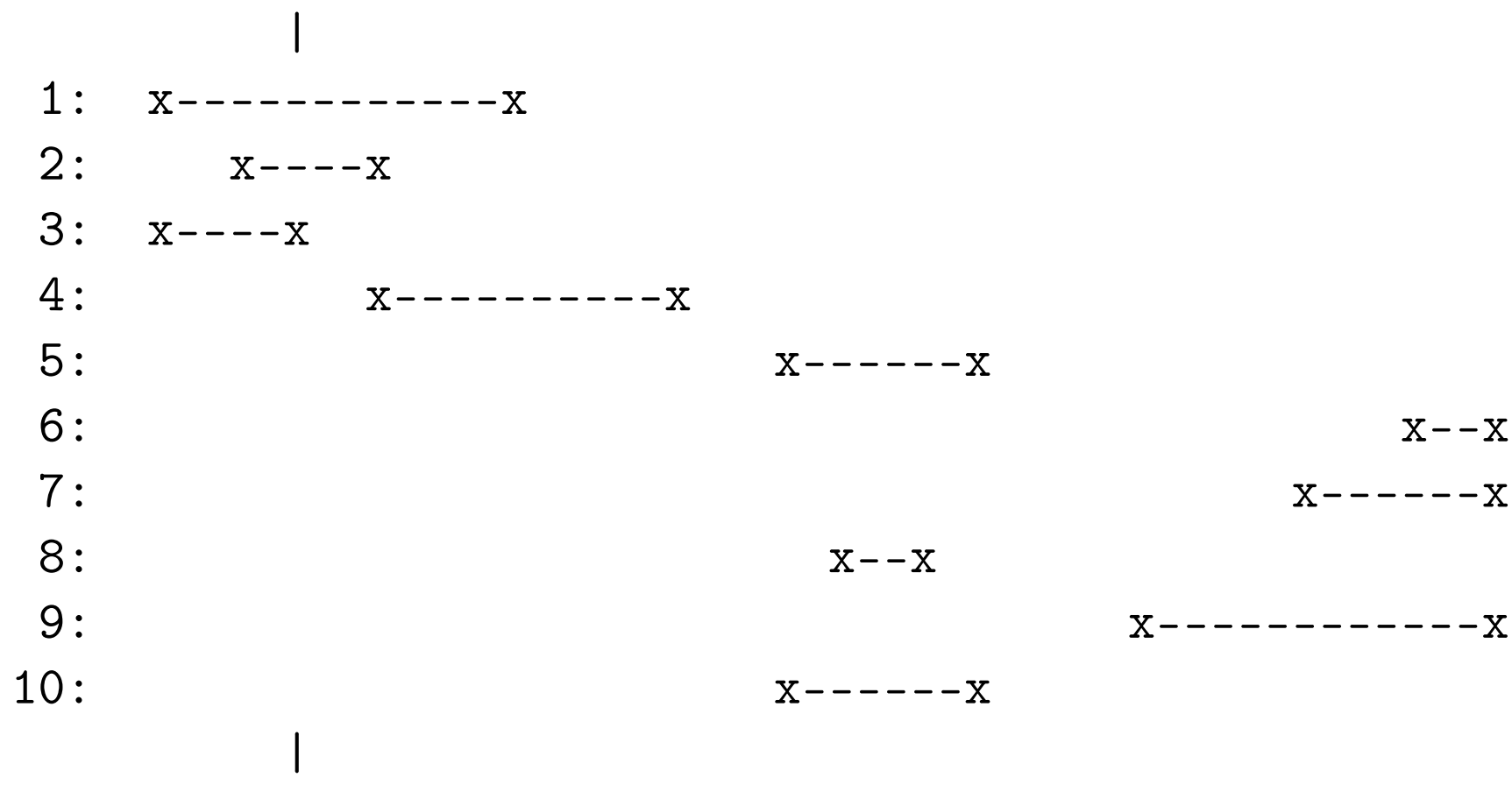
[1, 3]

r = 0


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

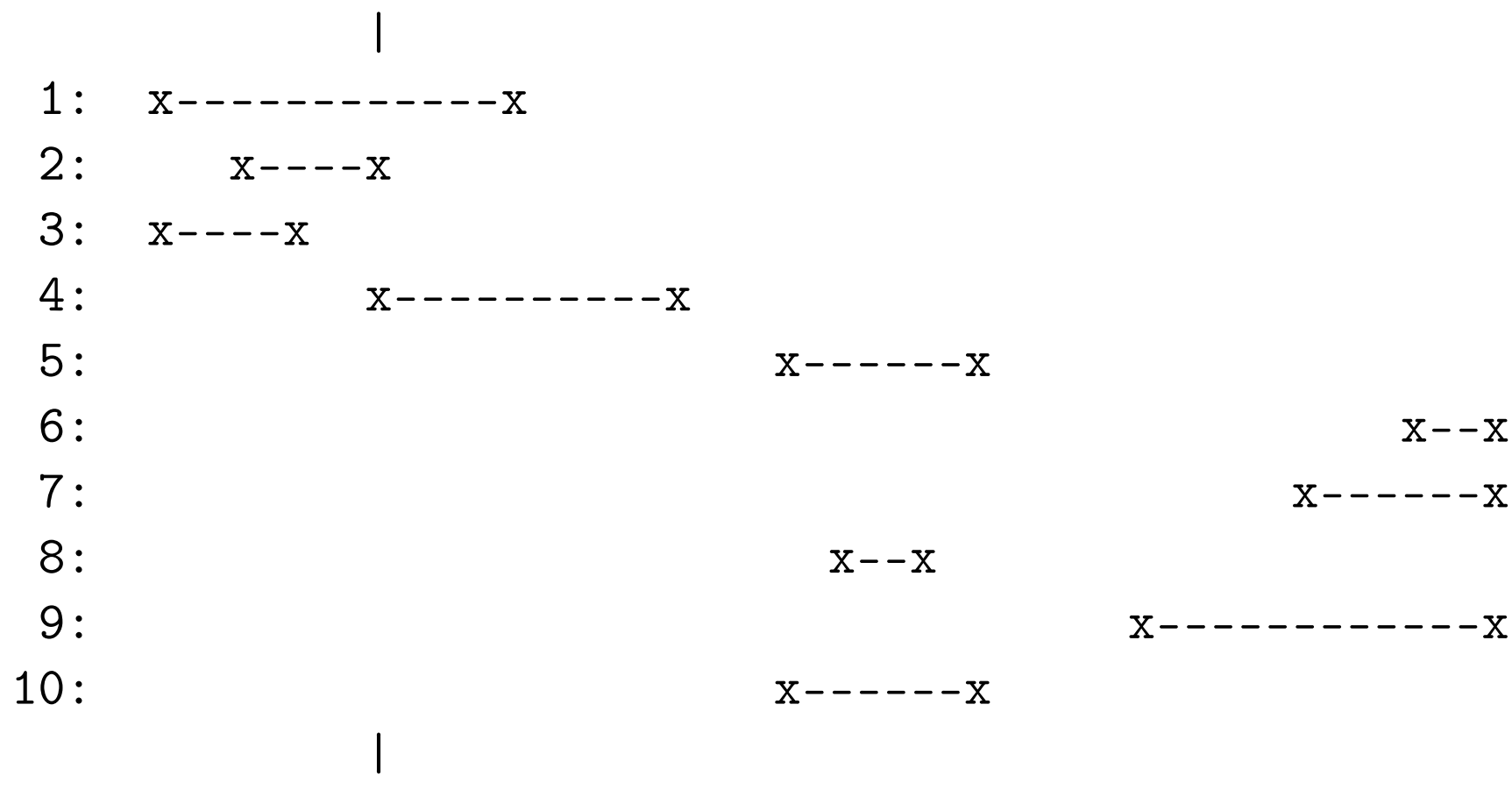


[1, 2, 3]
r = 0



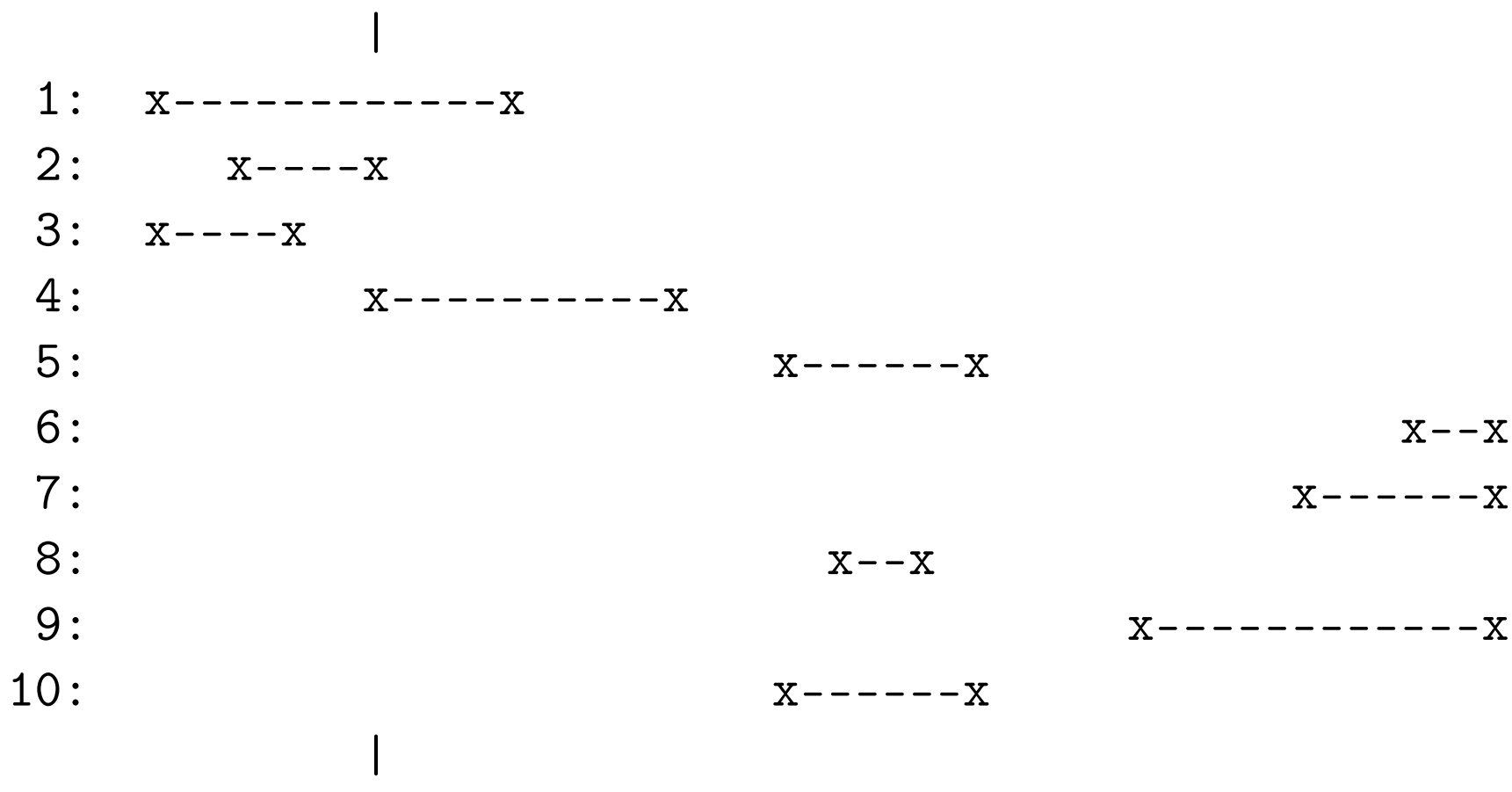
[1, 2]

r = 0



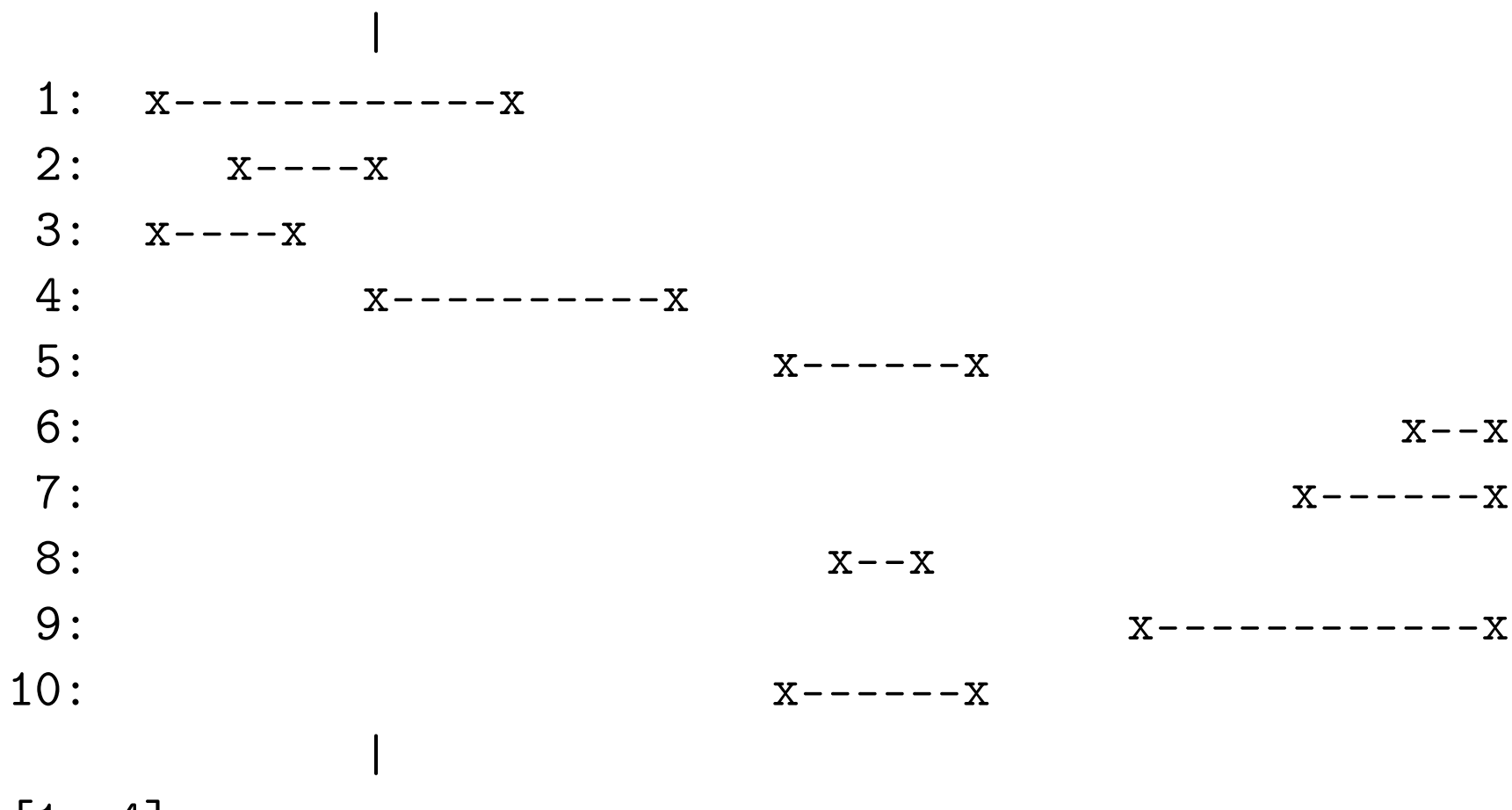
[1, 2]

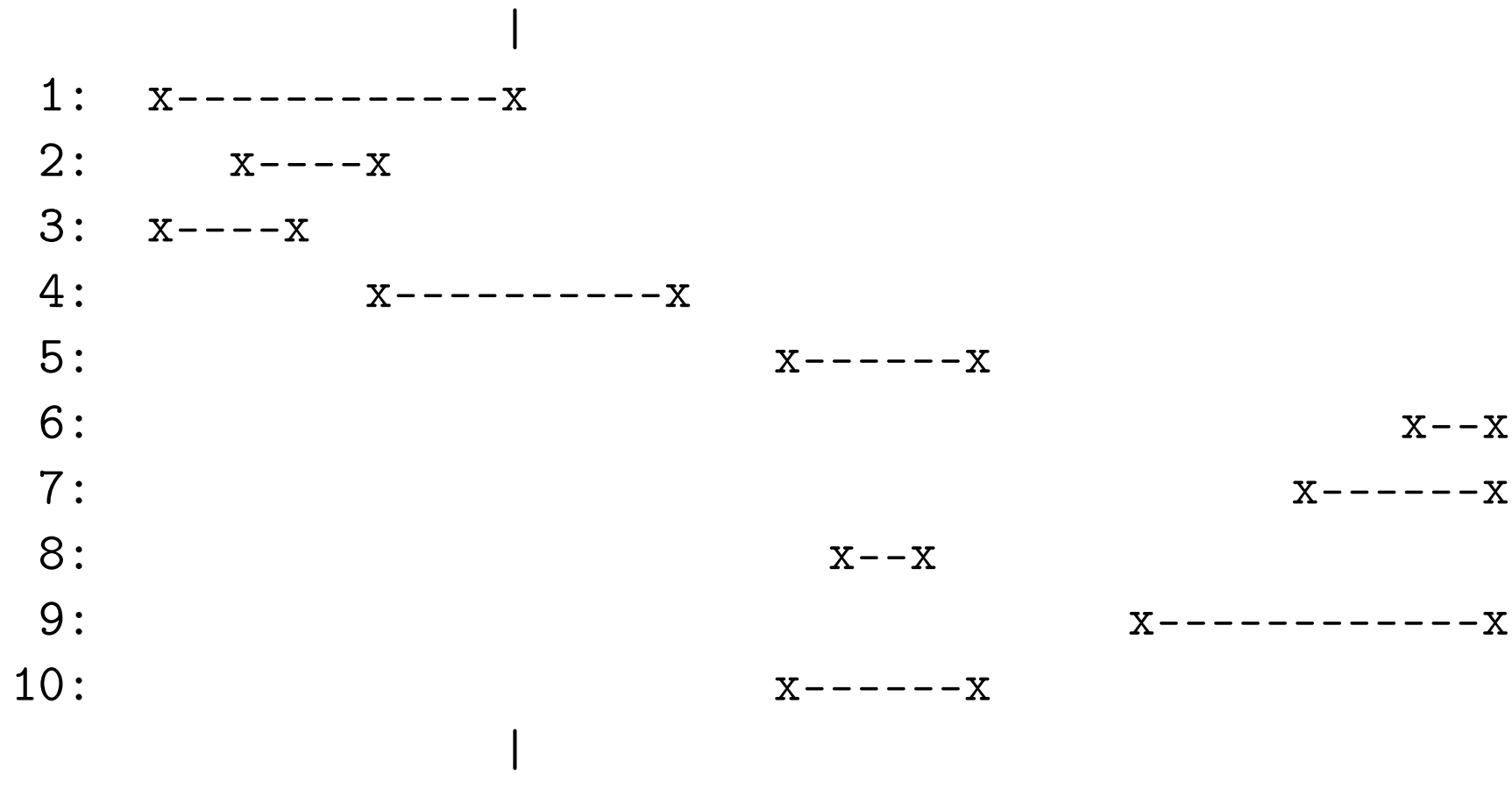
r = 0



[1, 2, 4]

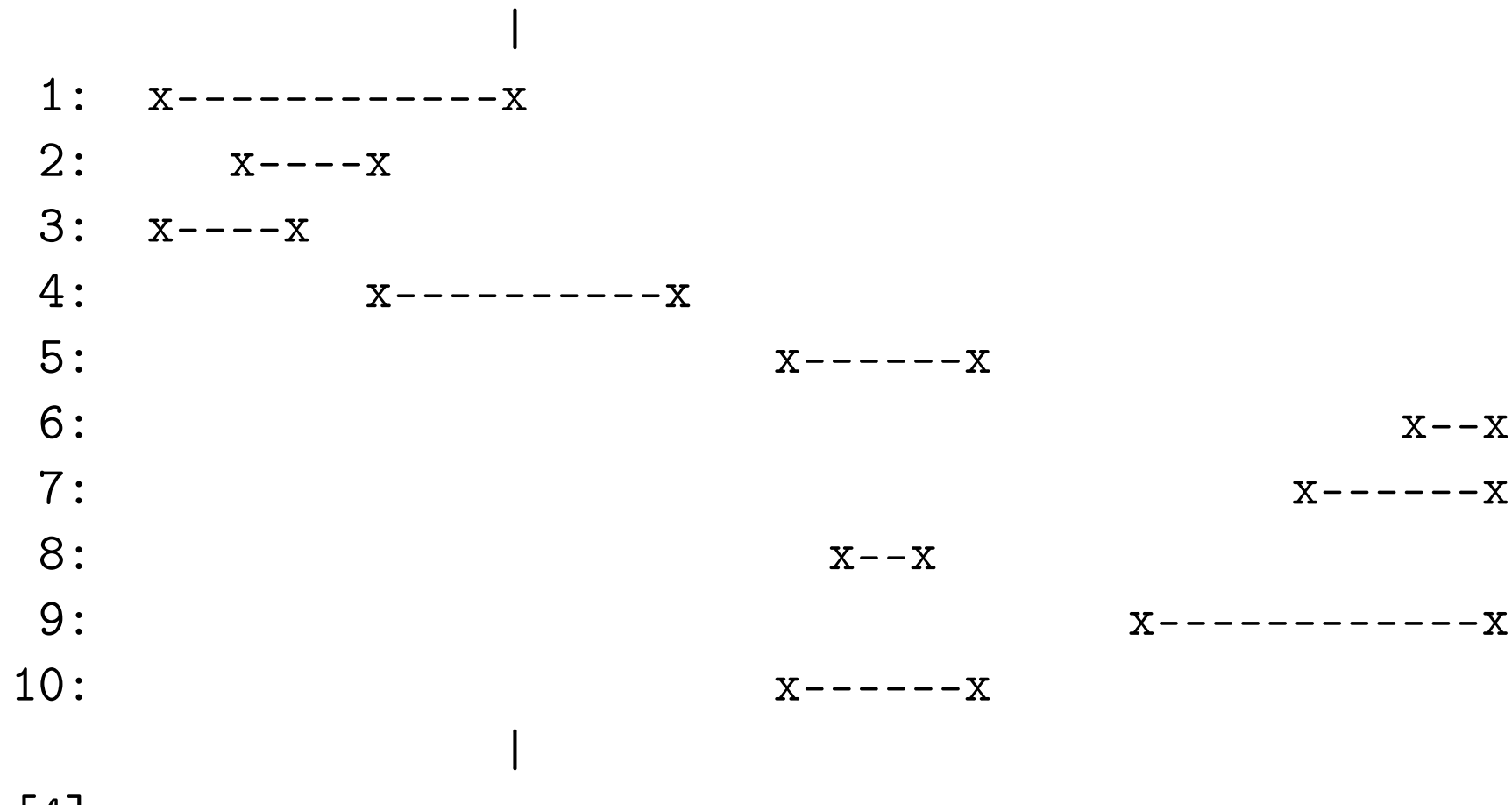
r = 0


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



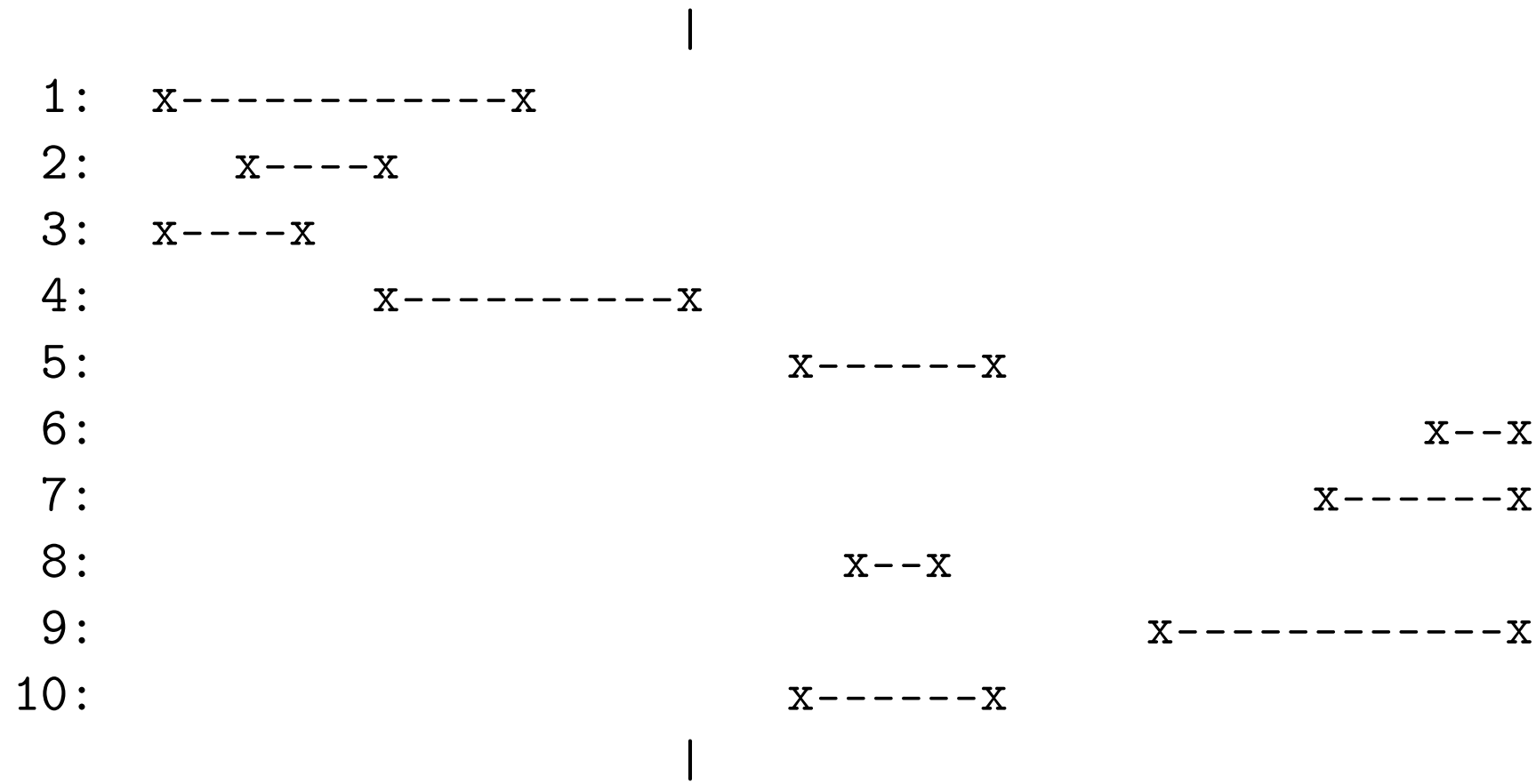
[1, 4]

r = 0



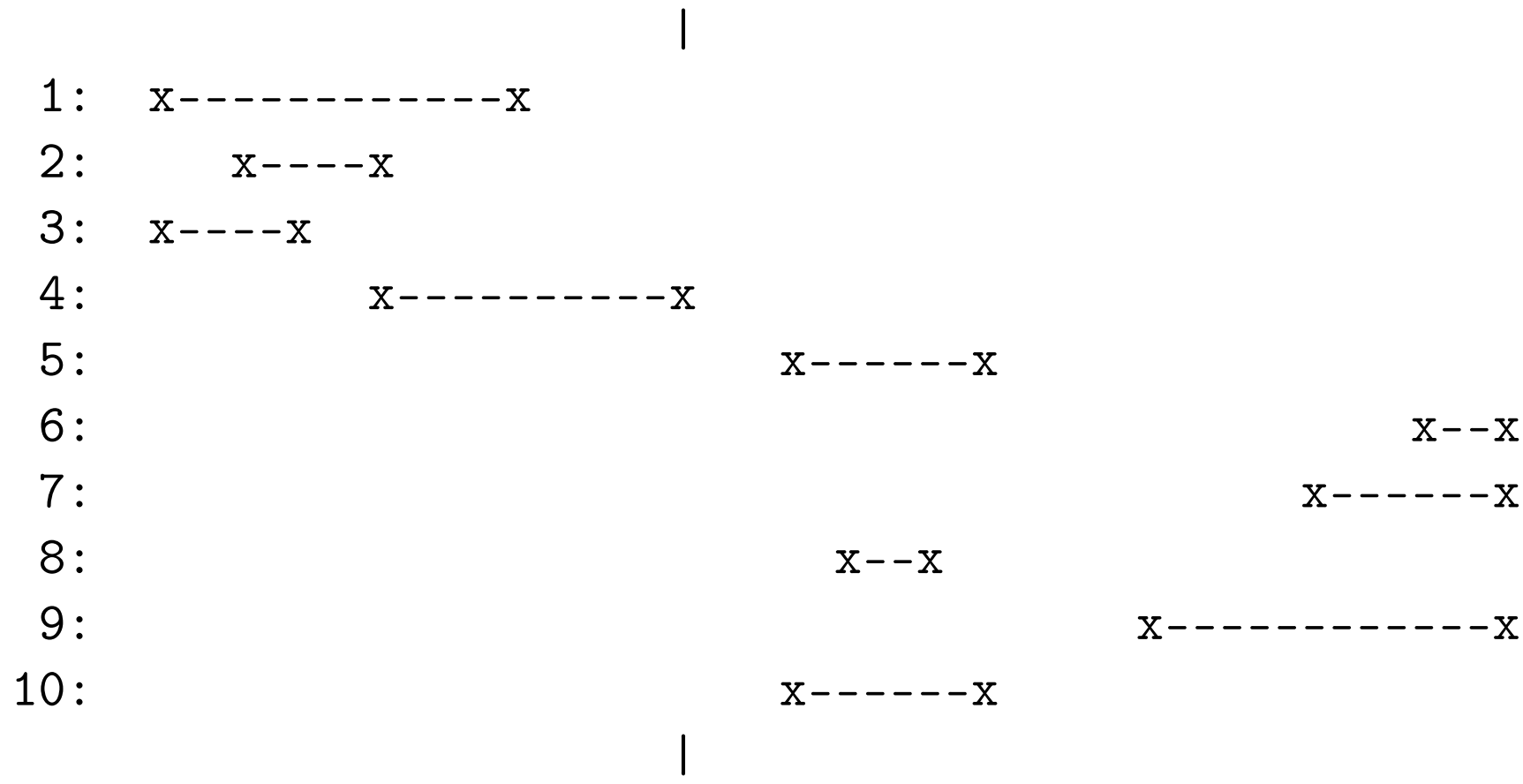
[4]

r = 0



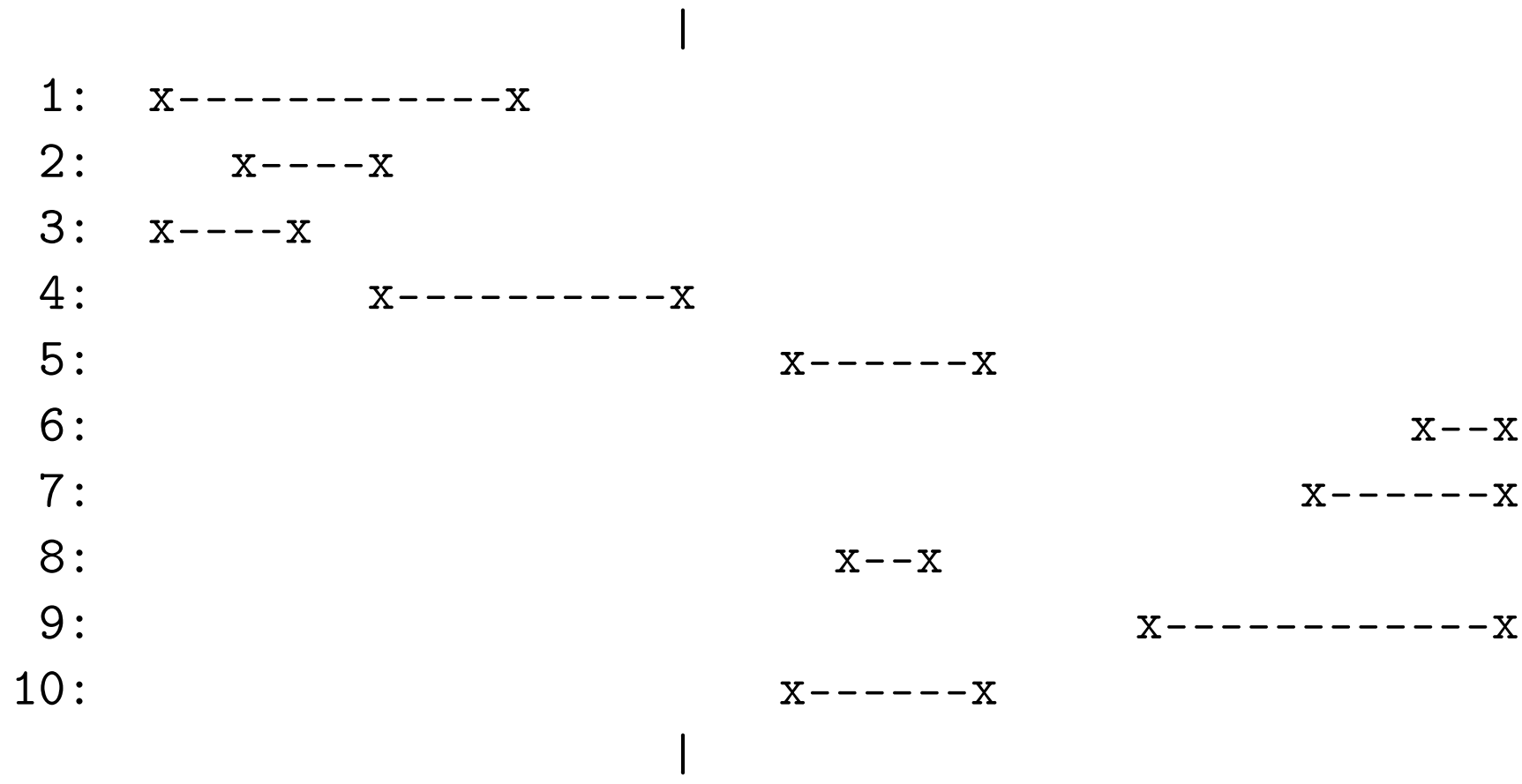
[4]

r = 0



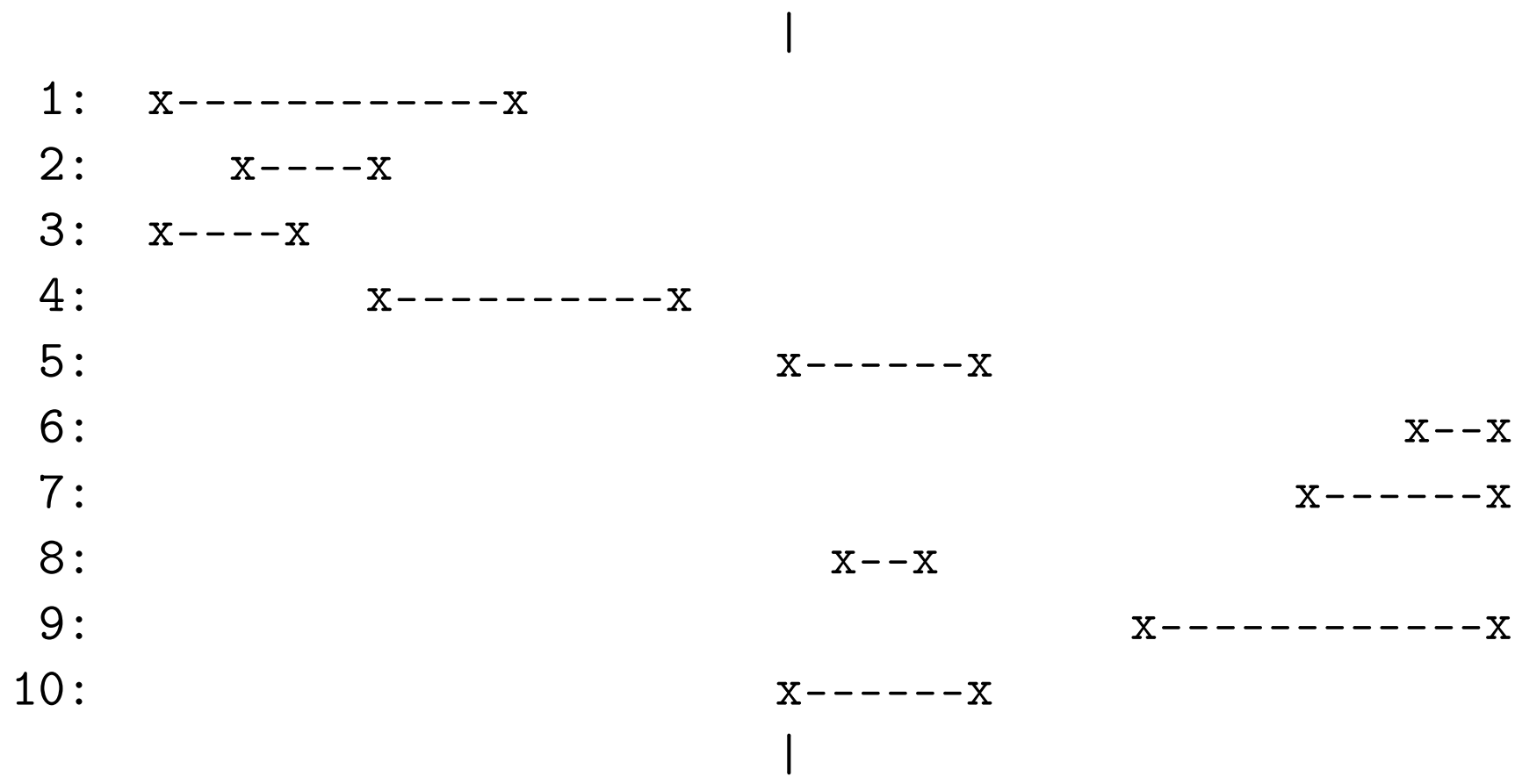
[]

r = 0



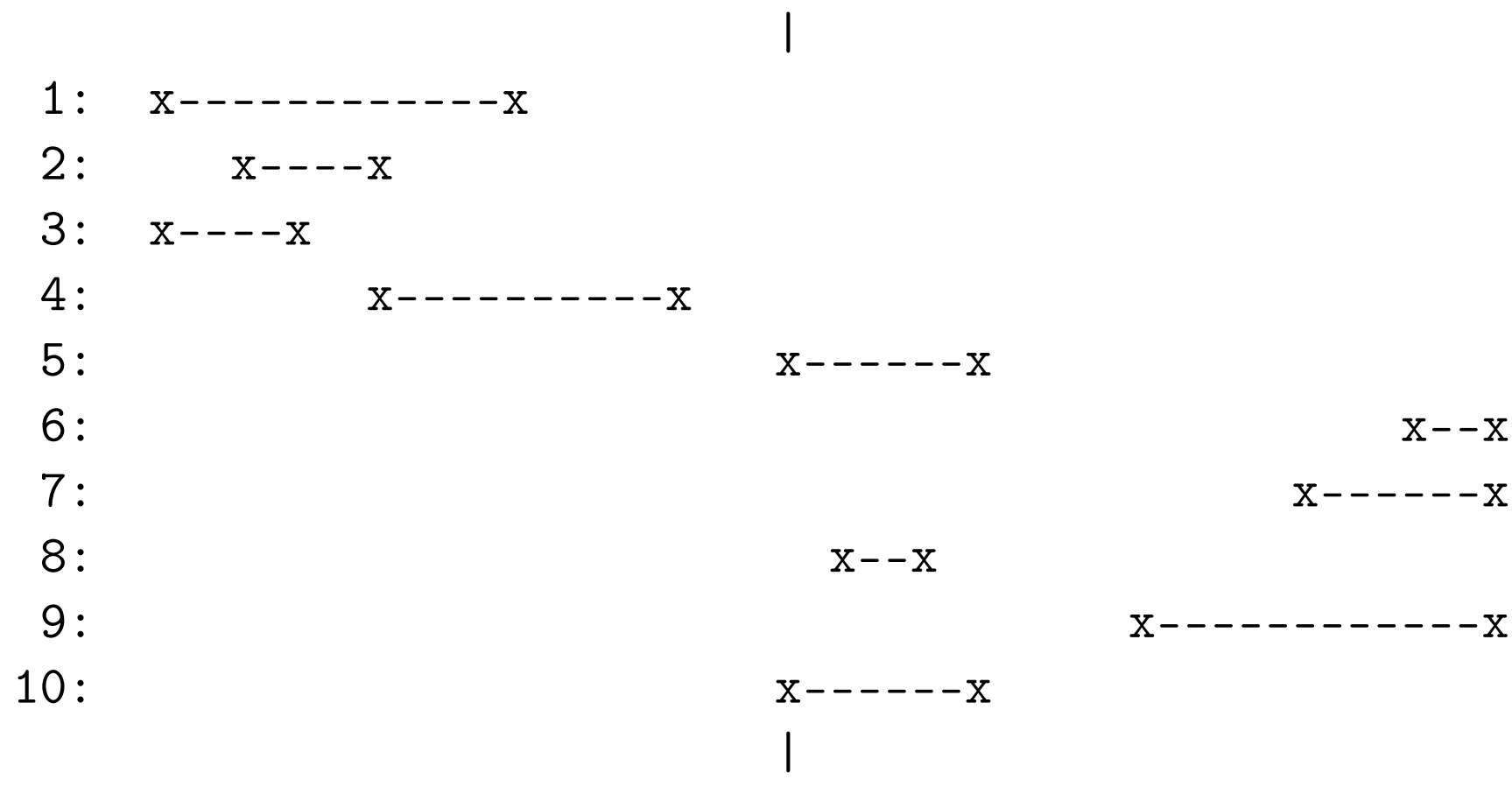
[]

r = 20



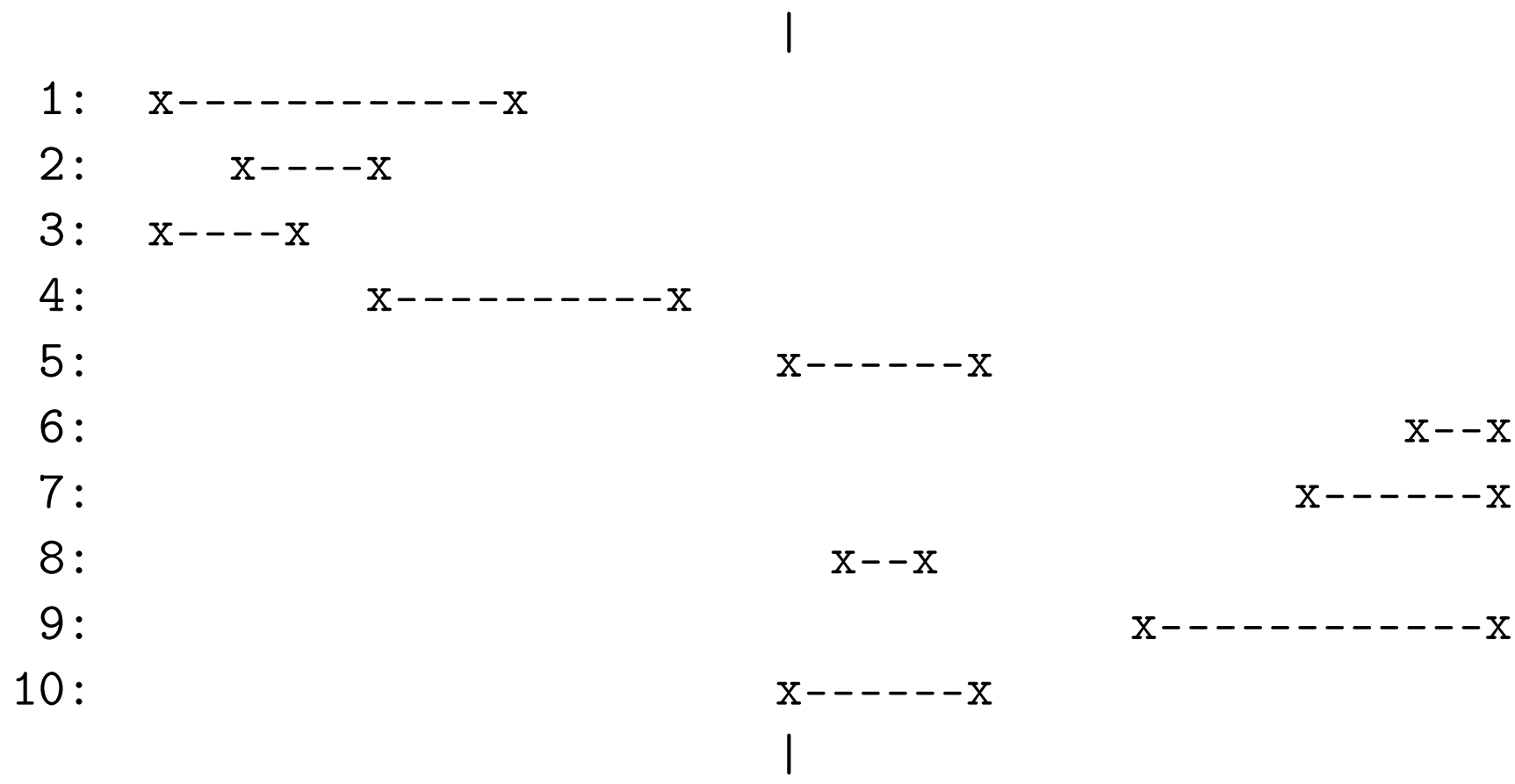
[]

r = 20

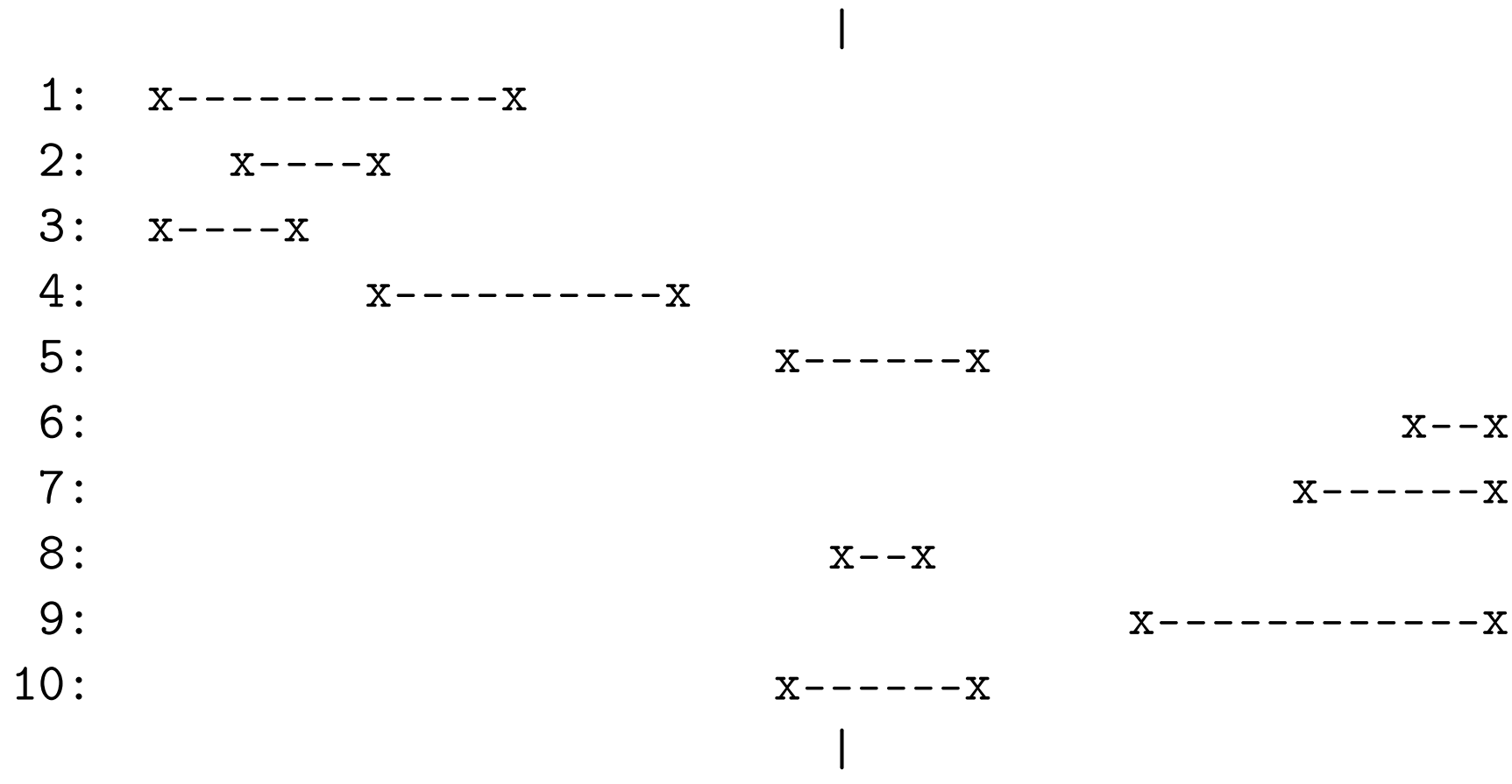


[5]

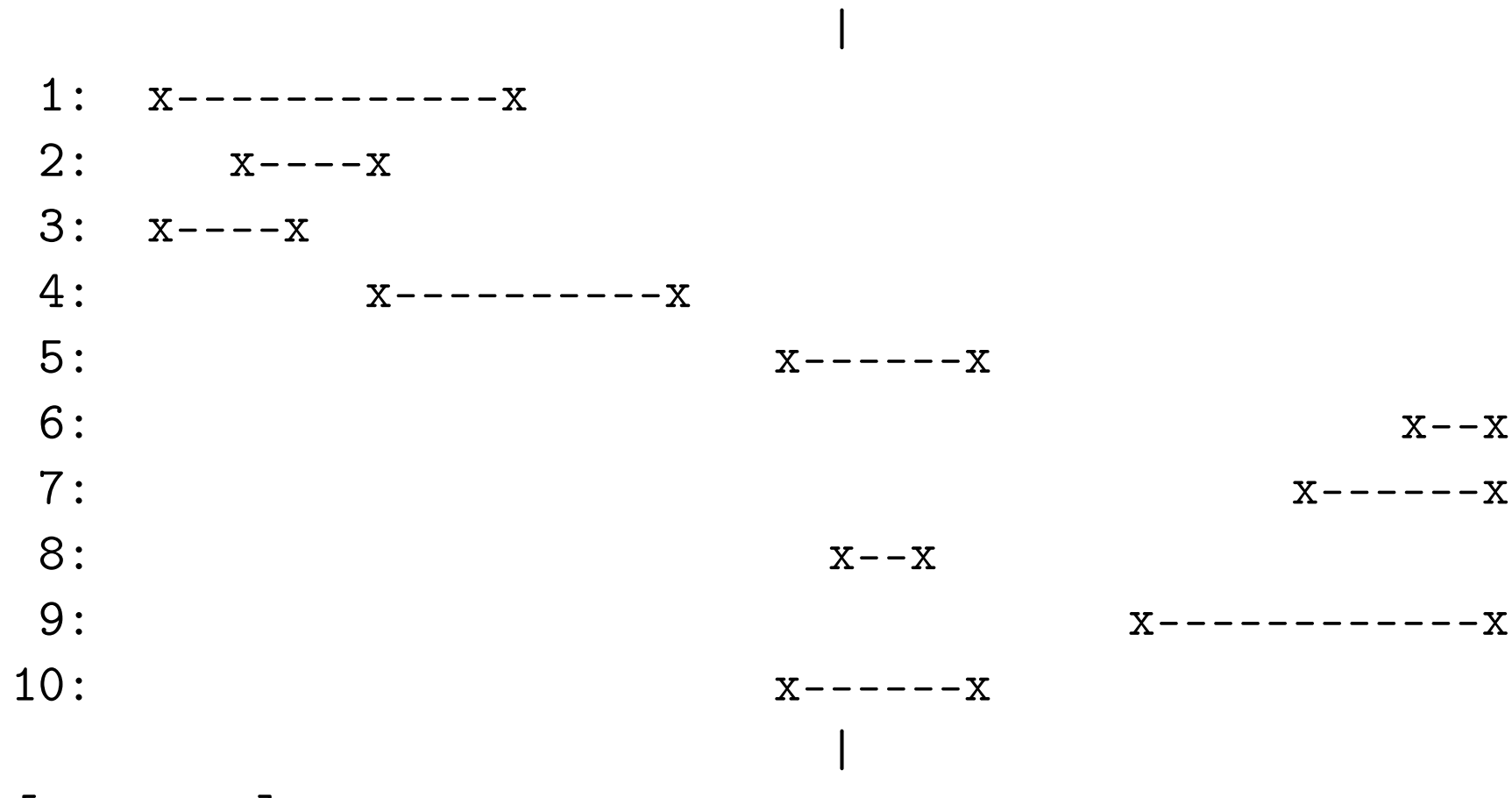
r = 20



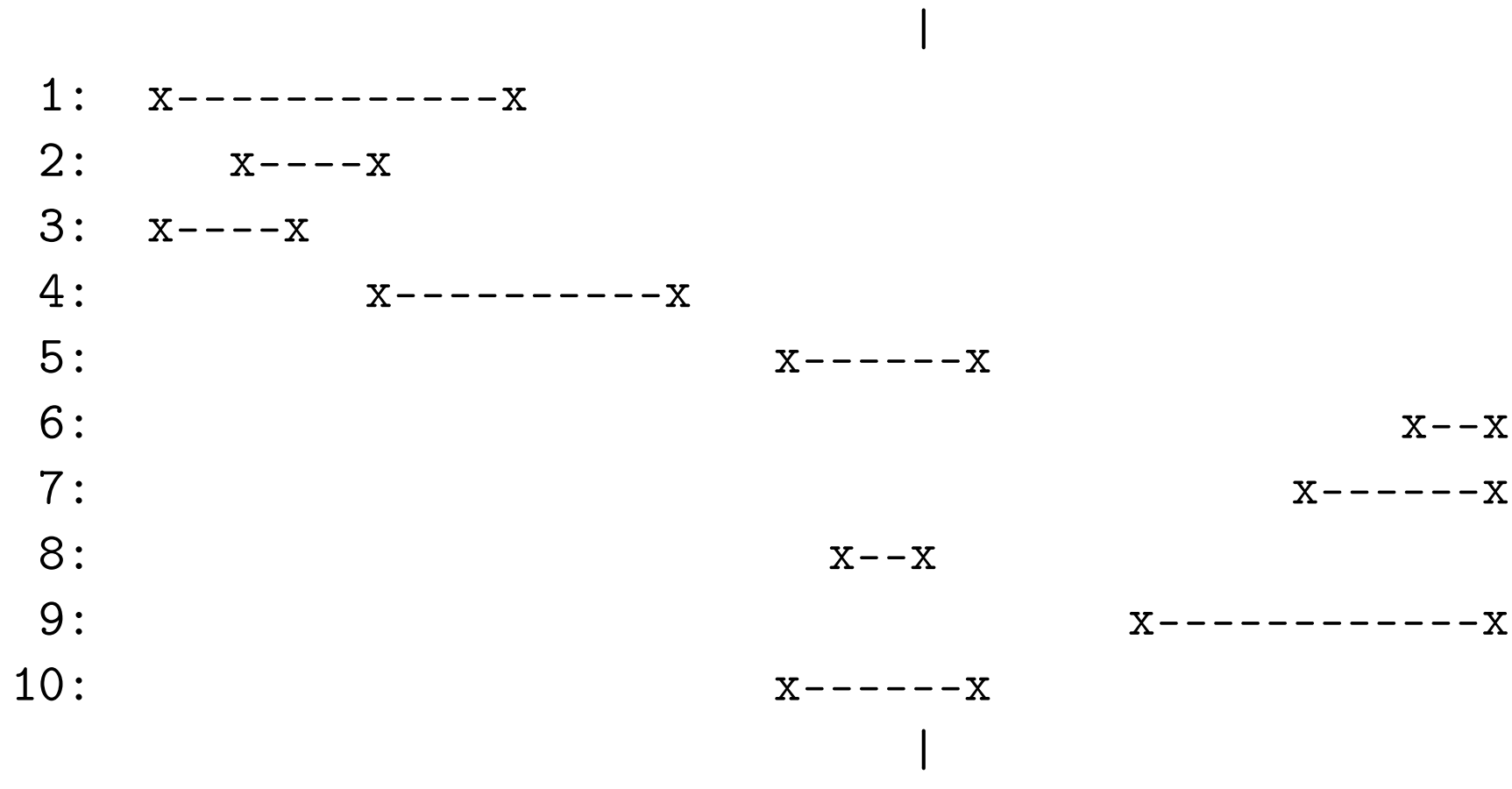
[5, 10]
r = 20



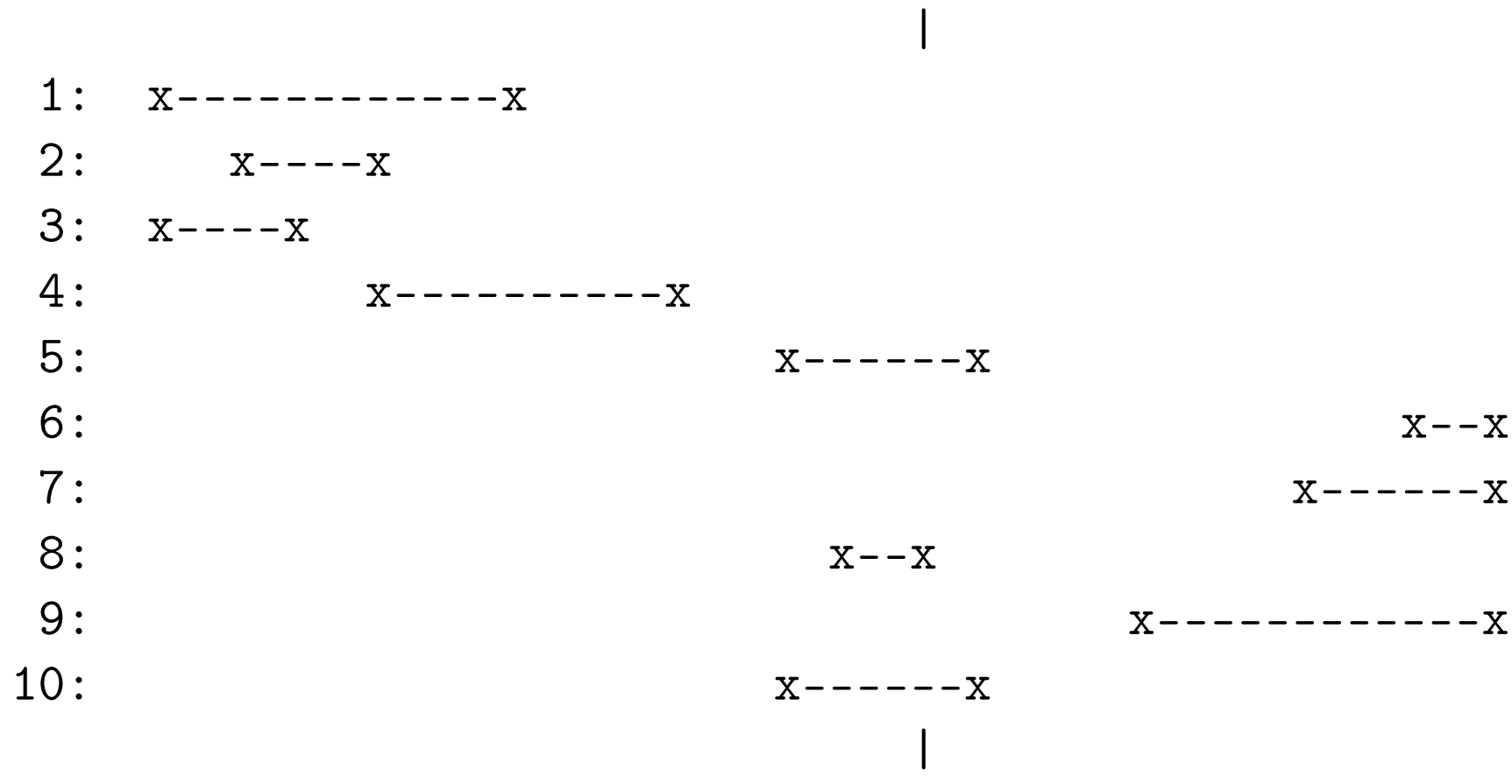
[5, 10]
r = 20



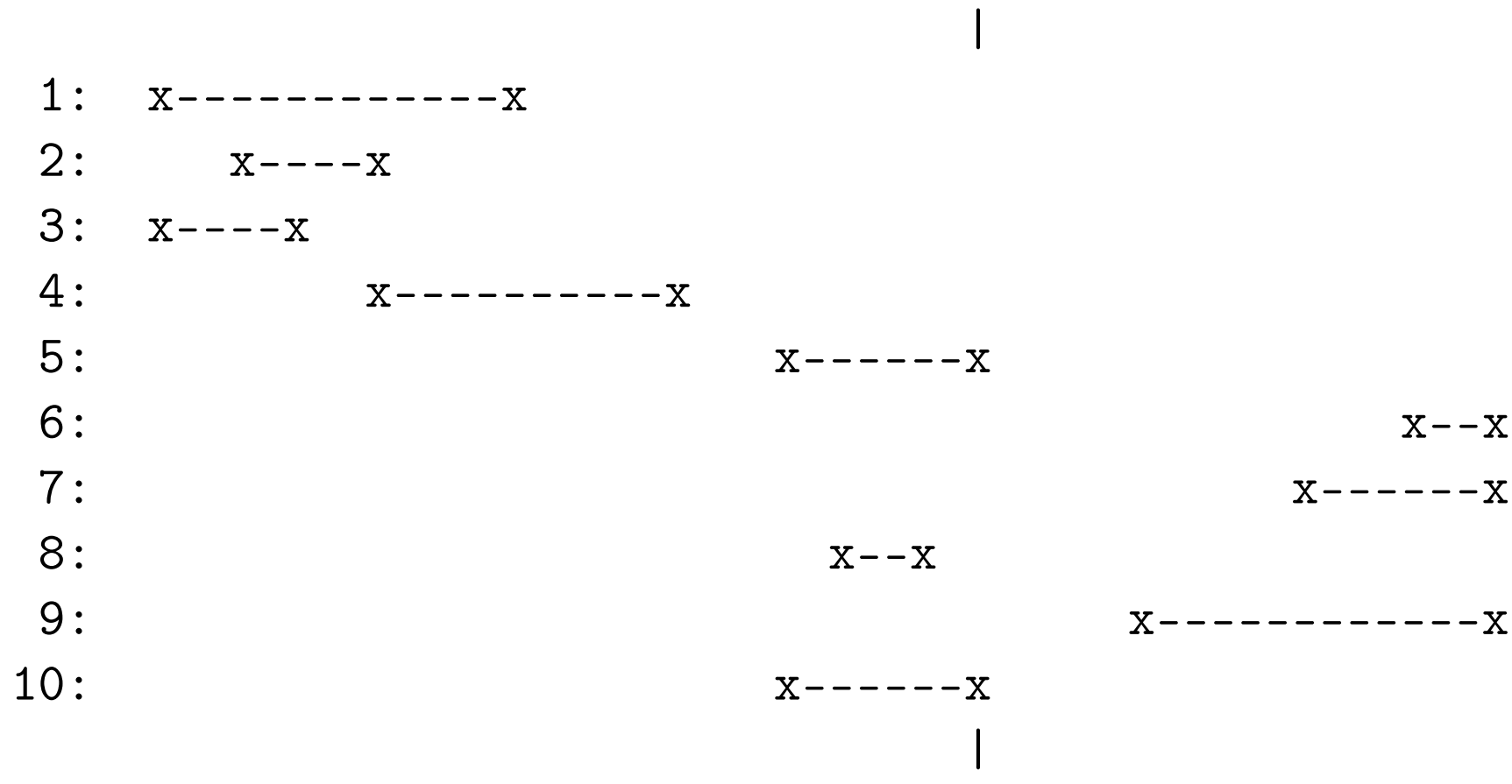
[5, 8, 10]
r = 20



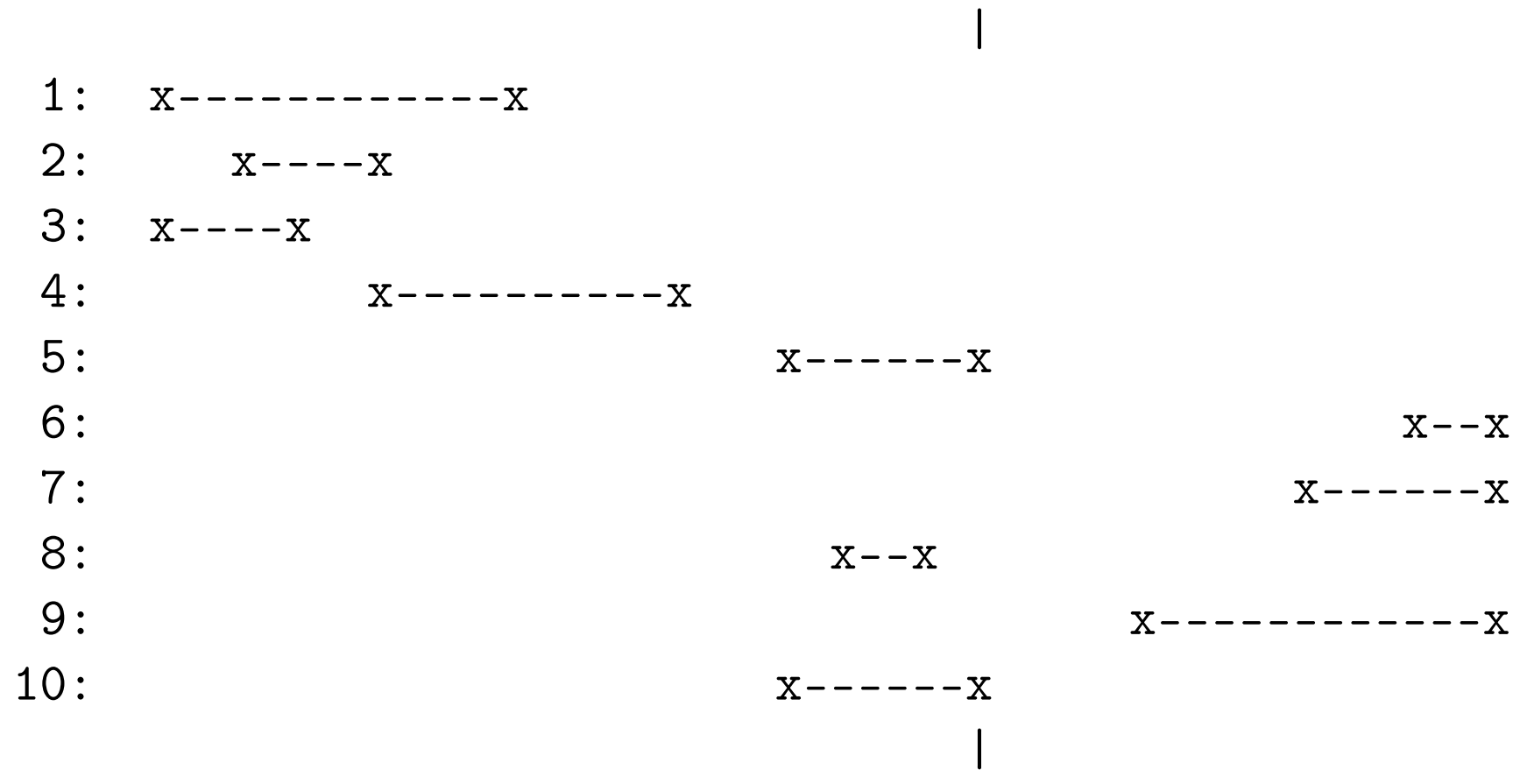
[5, 8, 10]
r = 20



[5, 10]
r = 20

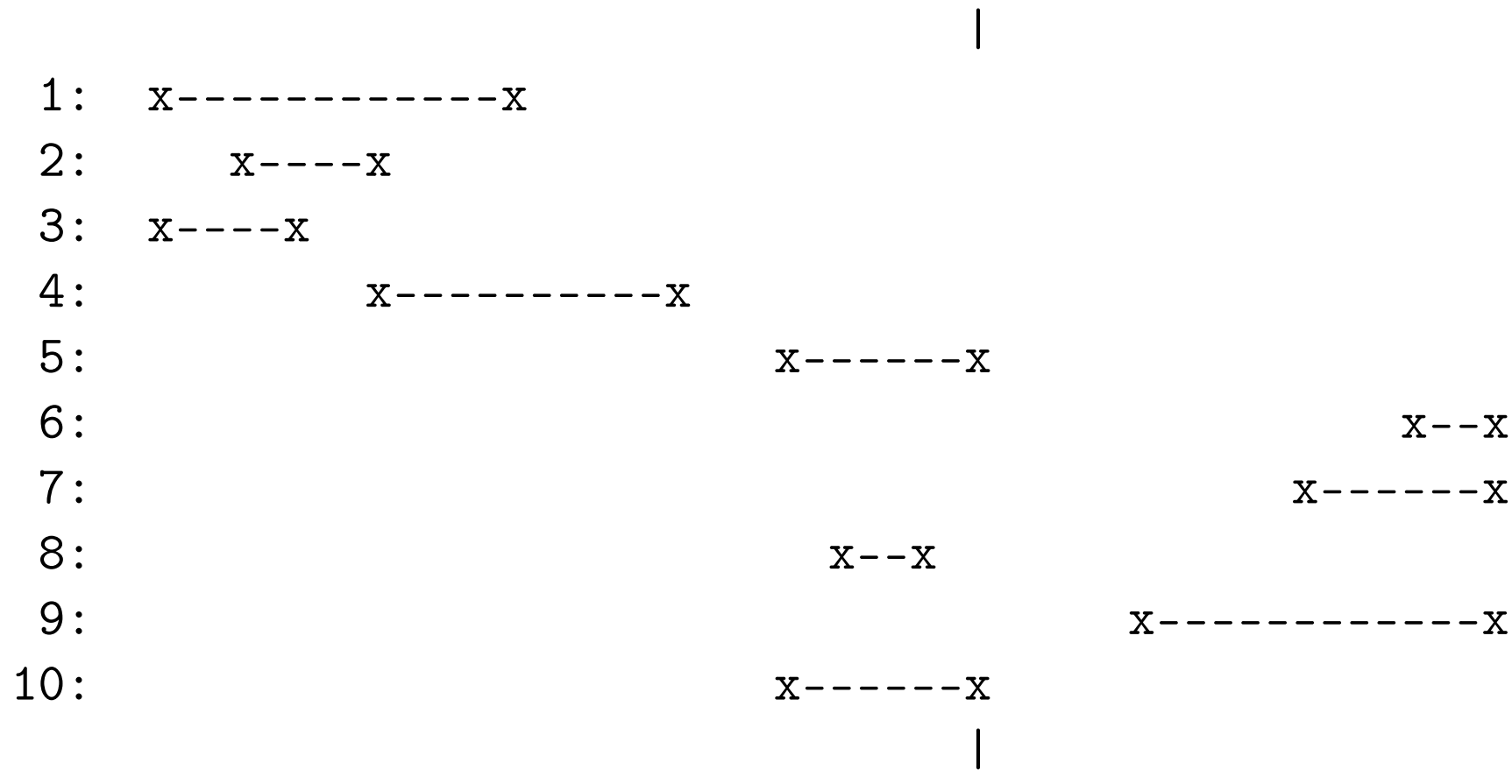


[5, 10]
r = 20



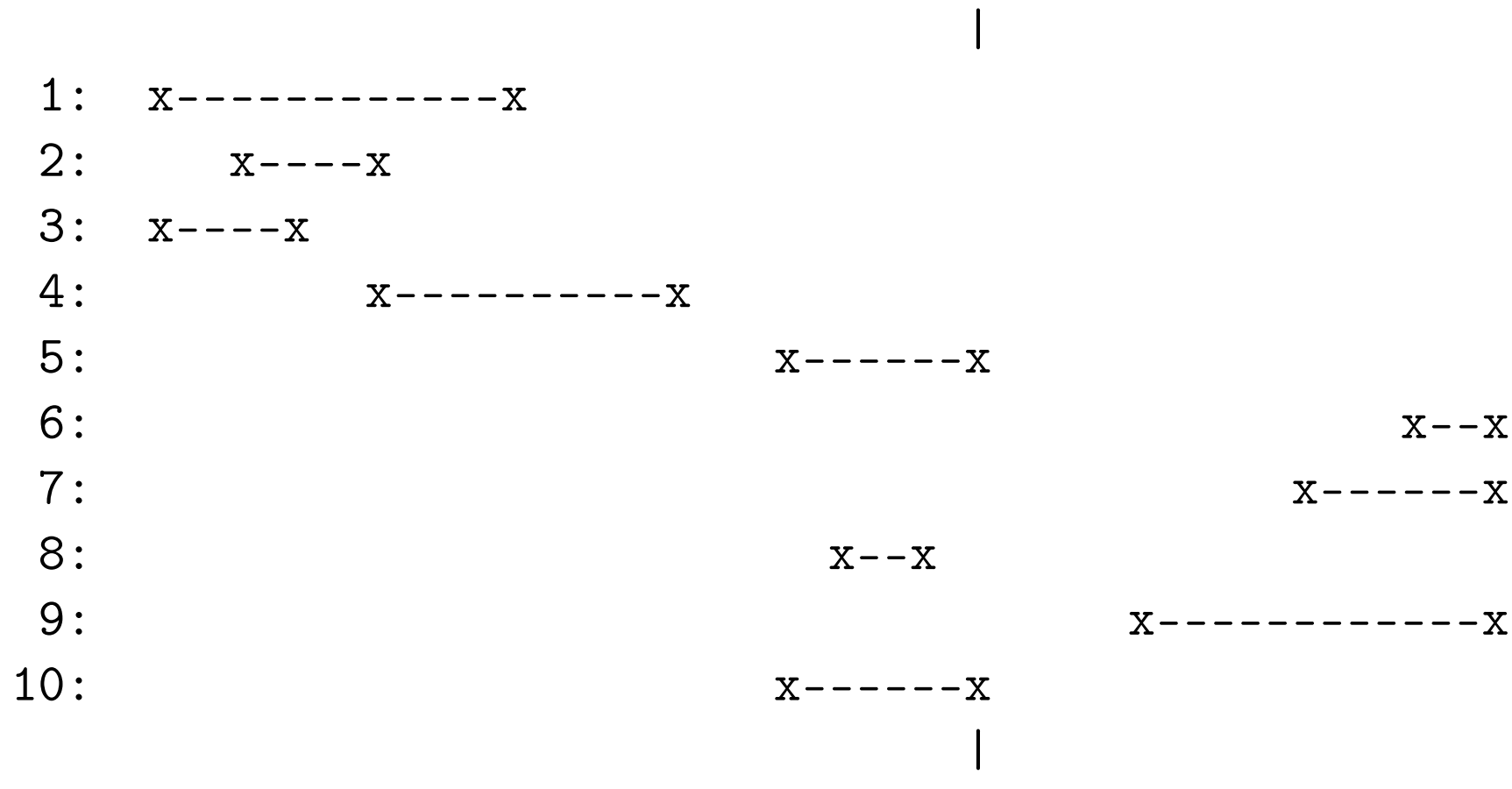
[5]

r = 20



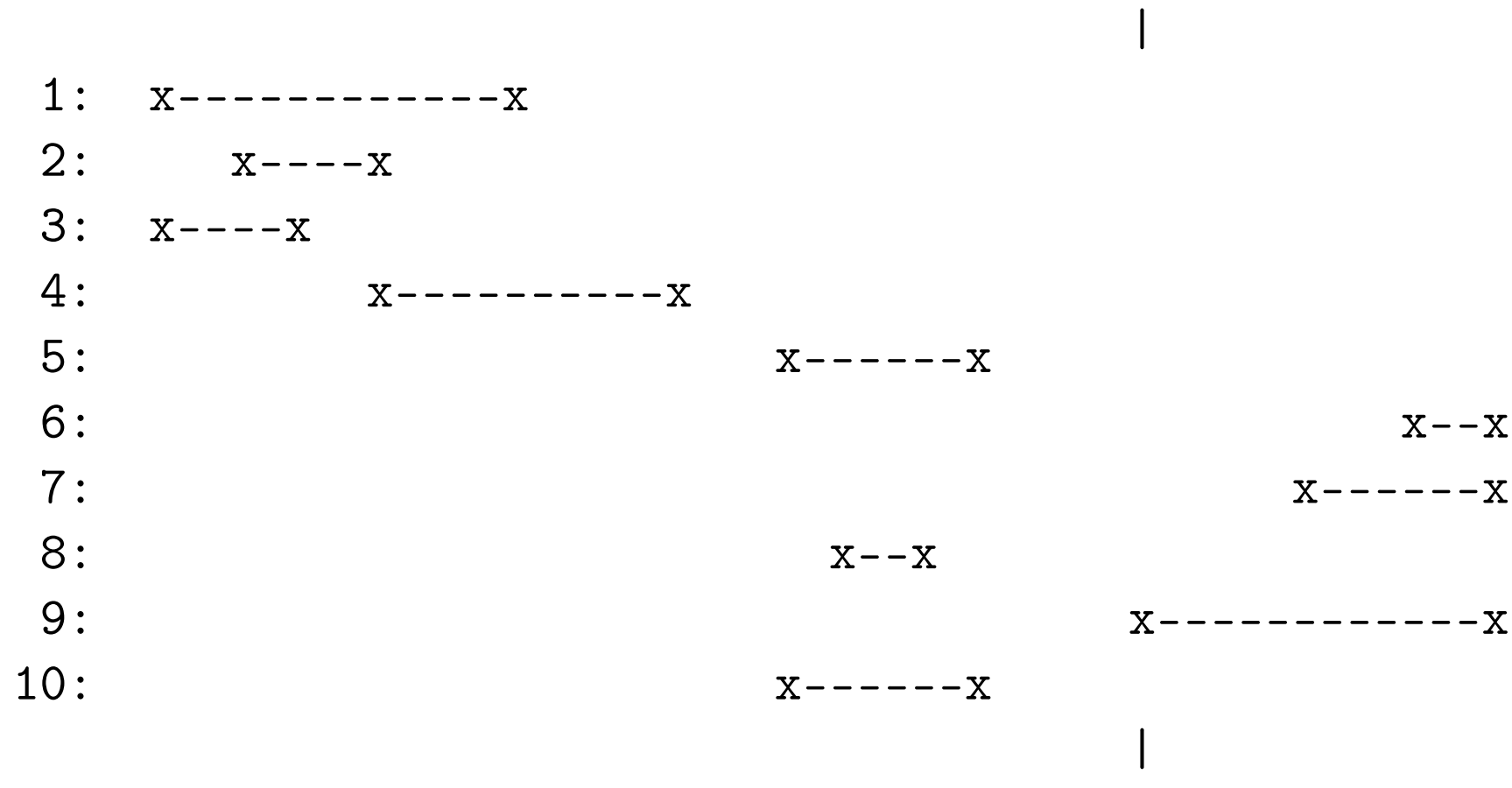
[]

r = 20



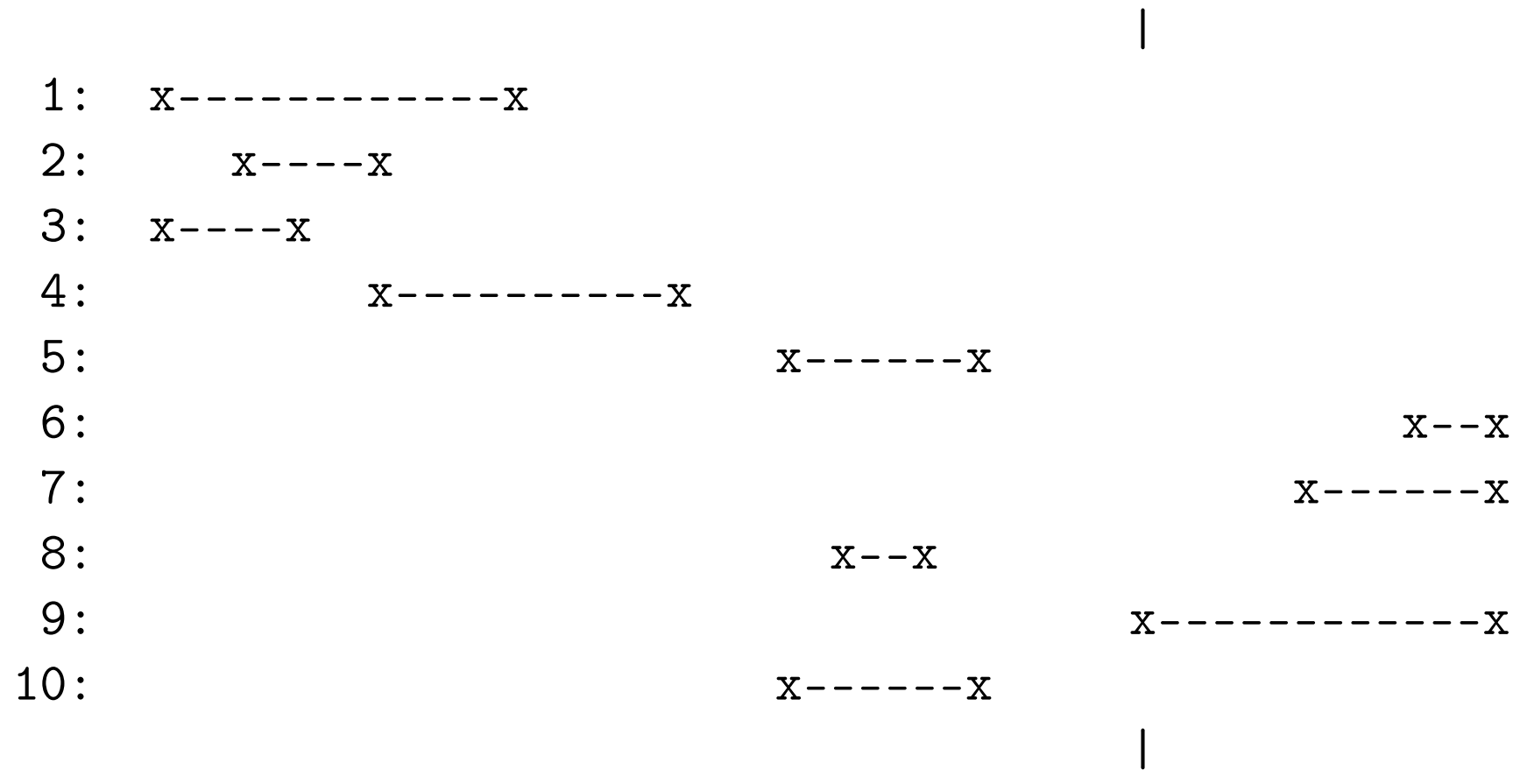
[]

r = 28



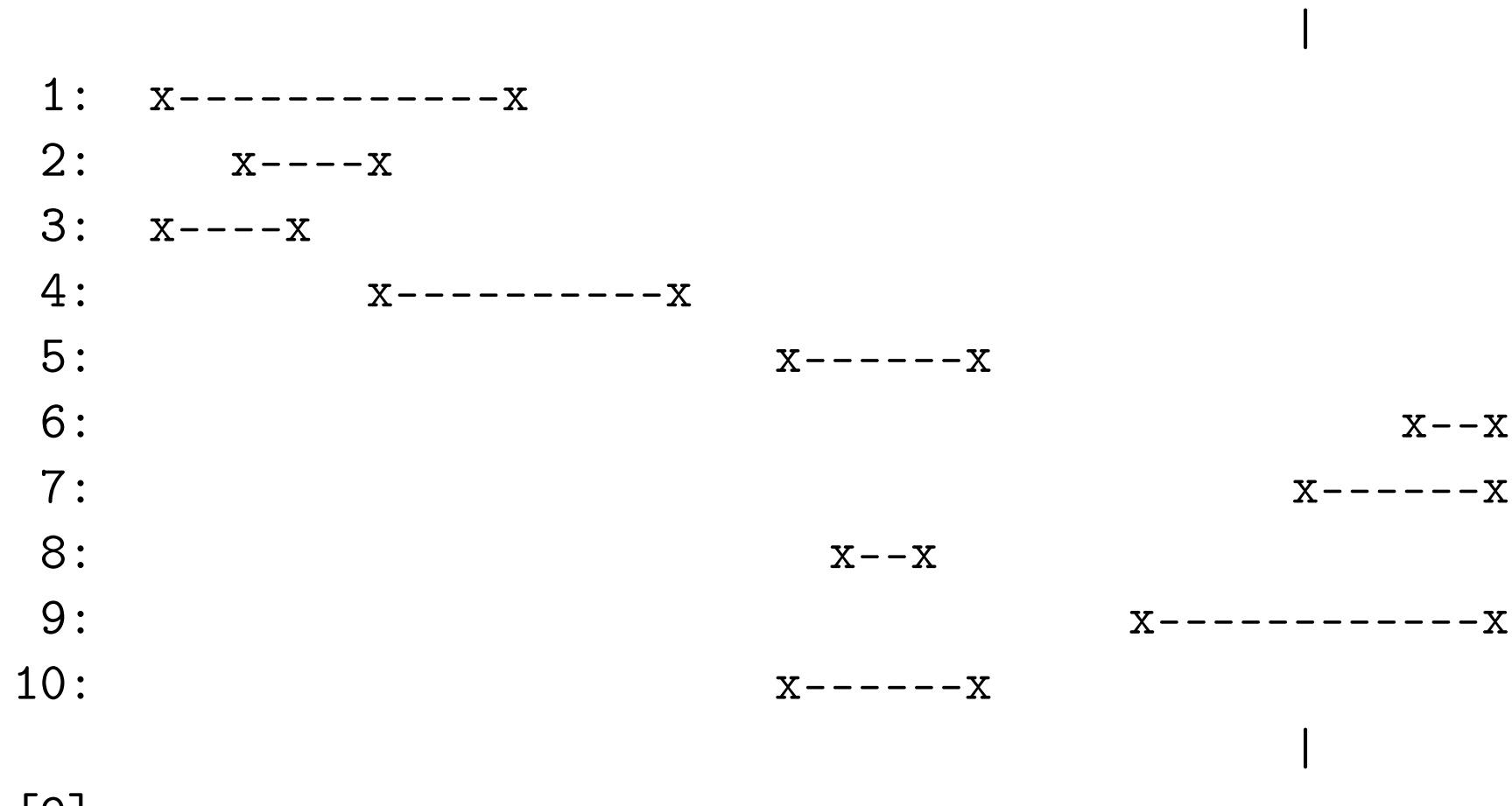
[]

r = 28



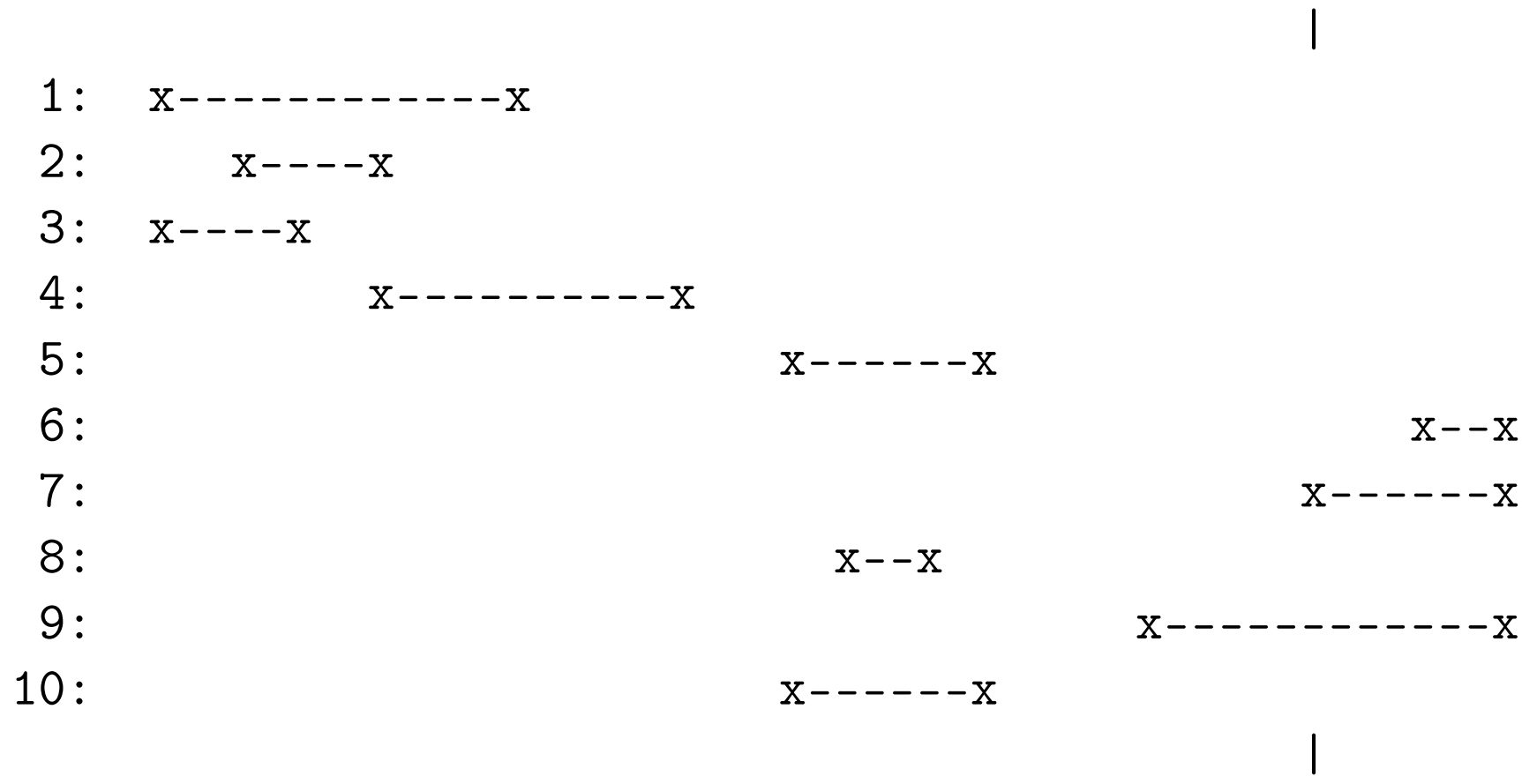
[9]

r = 28

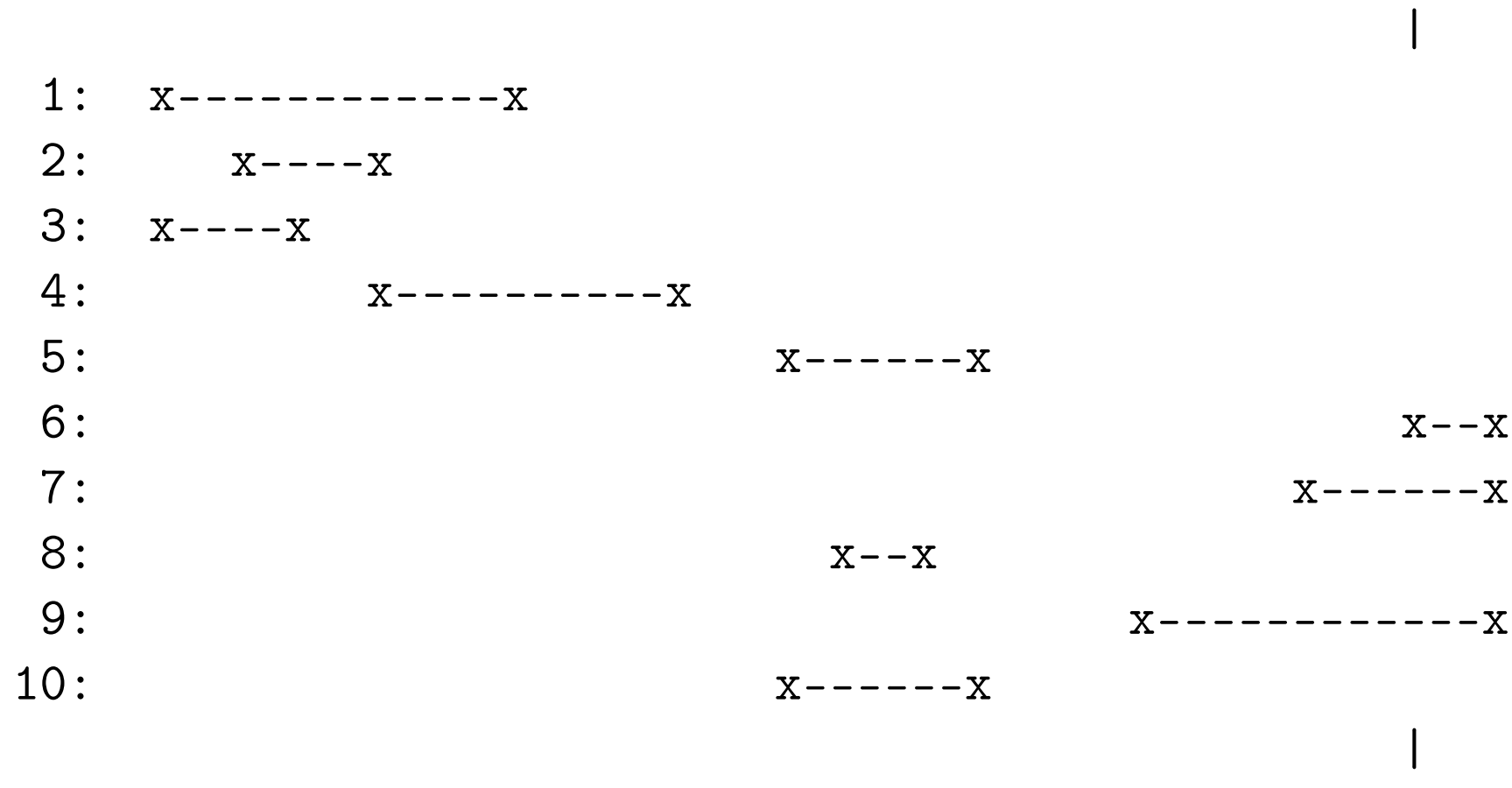


[9]

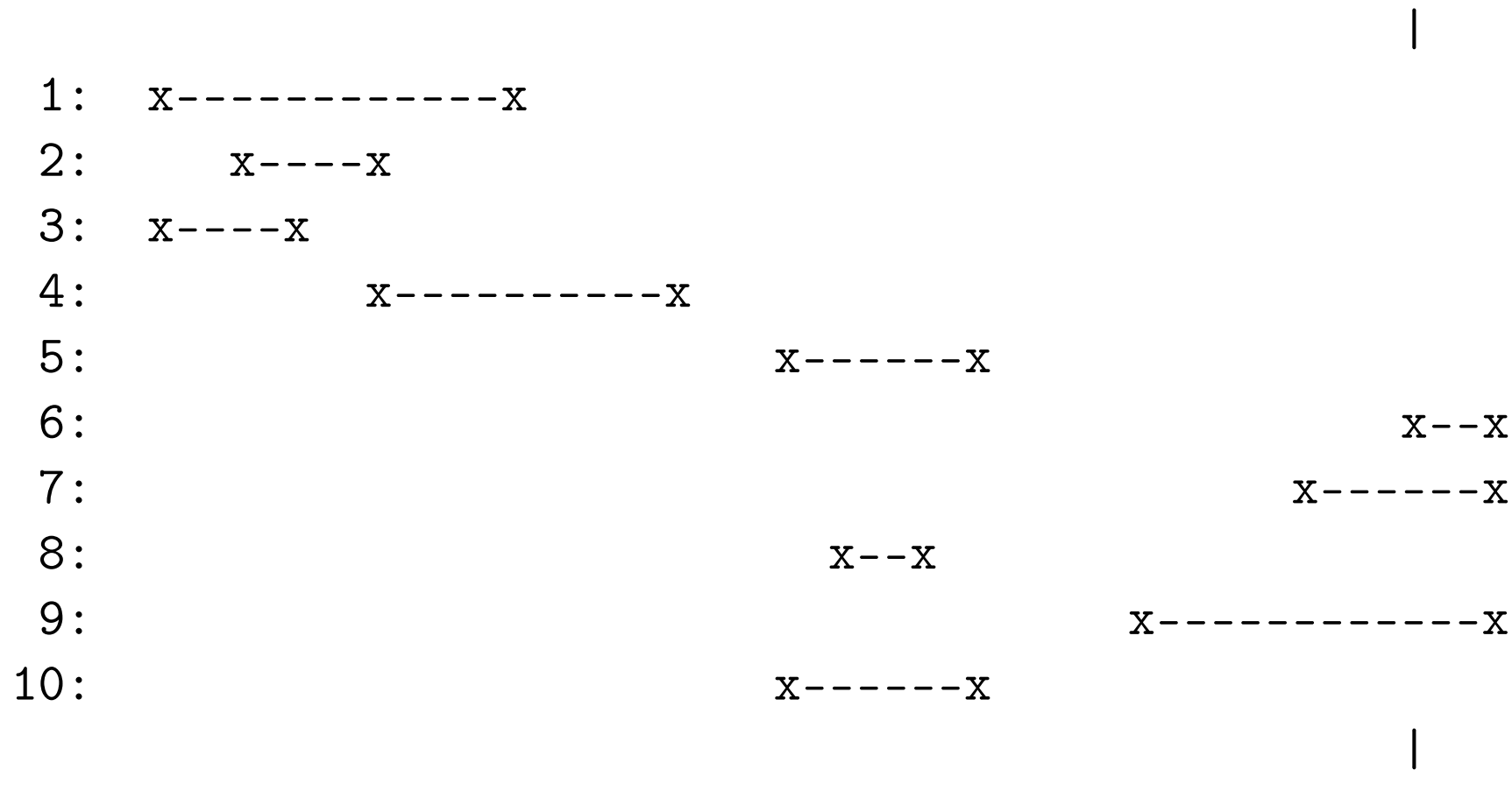
r = 28



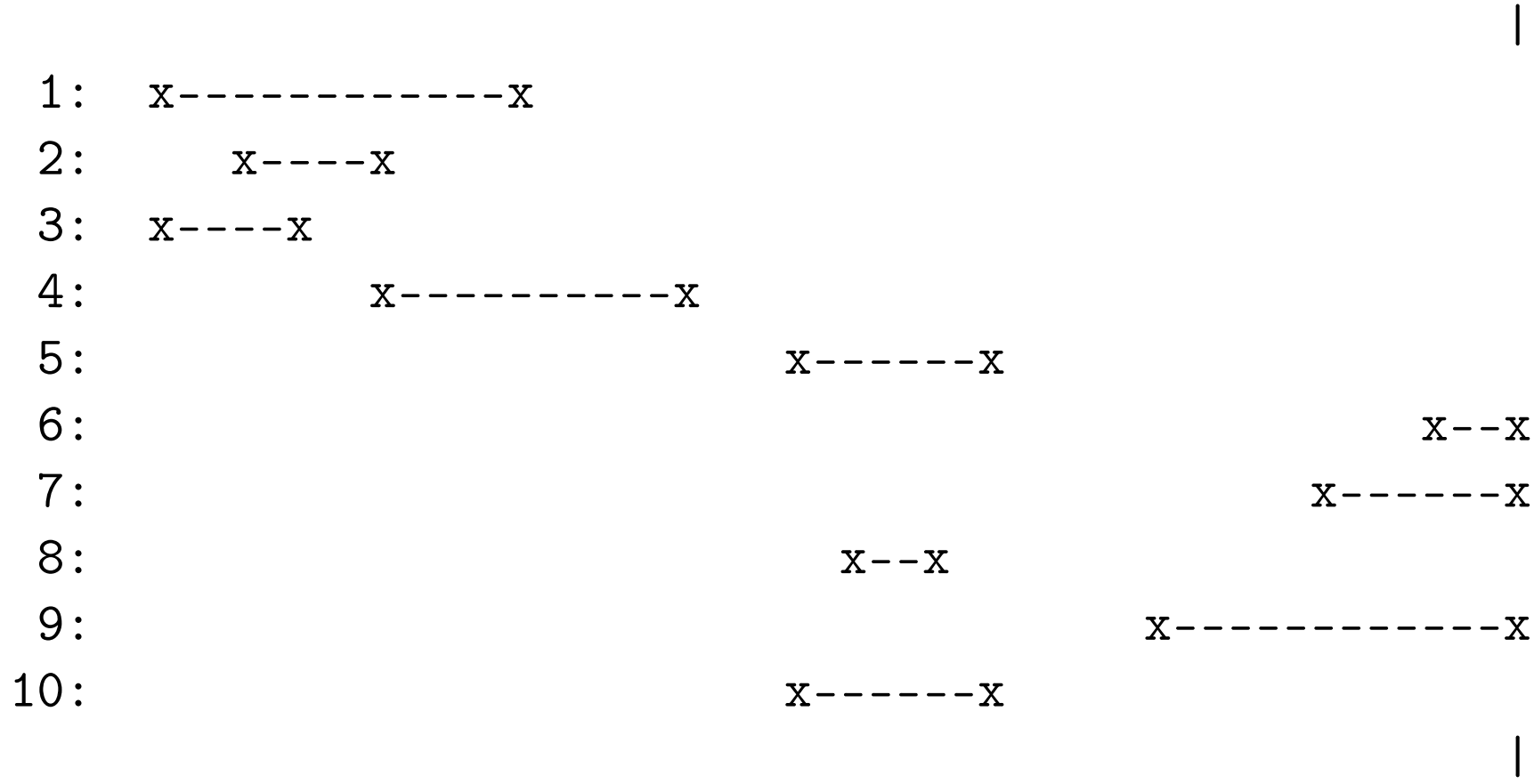
[7, 9]
r = 28



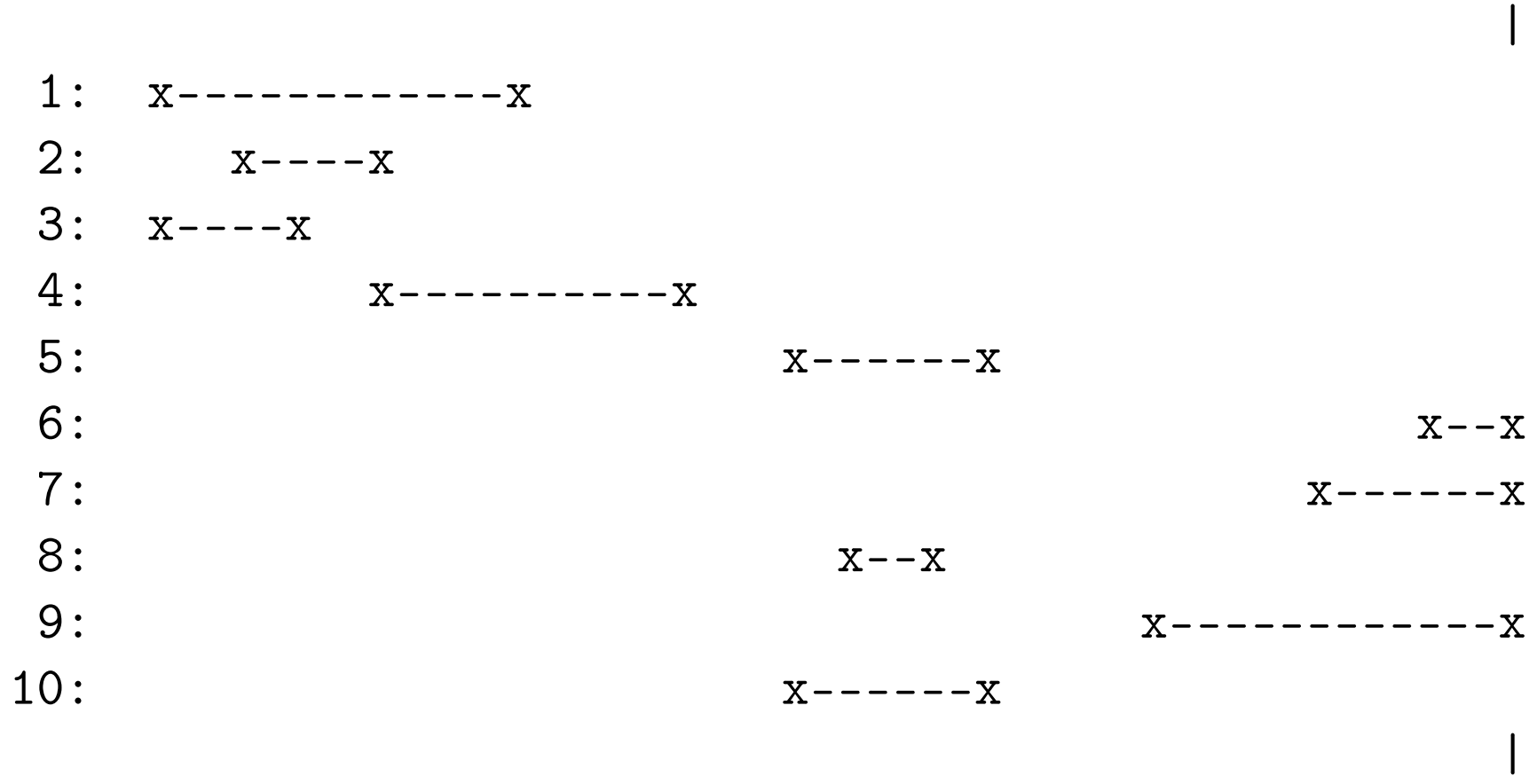
[7, 9]
r = 28



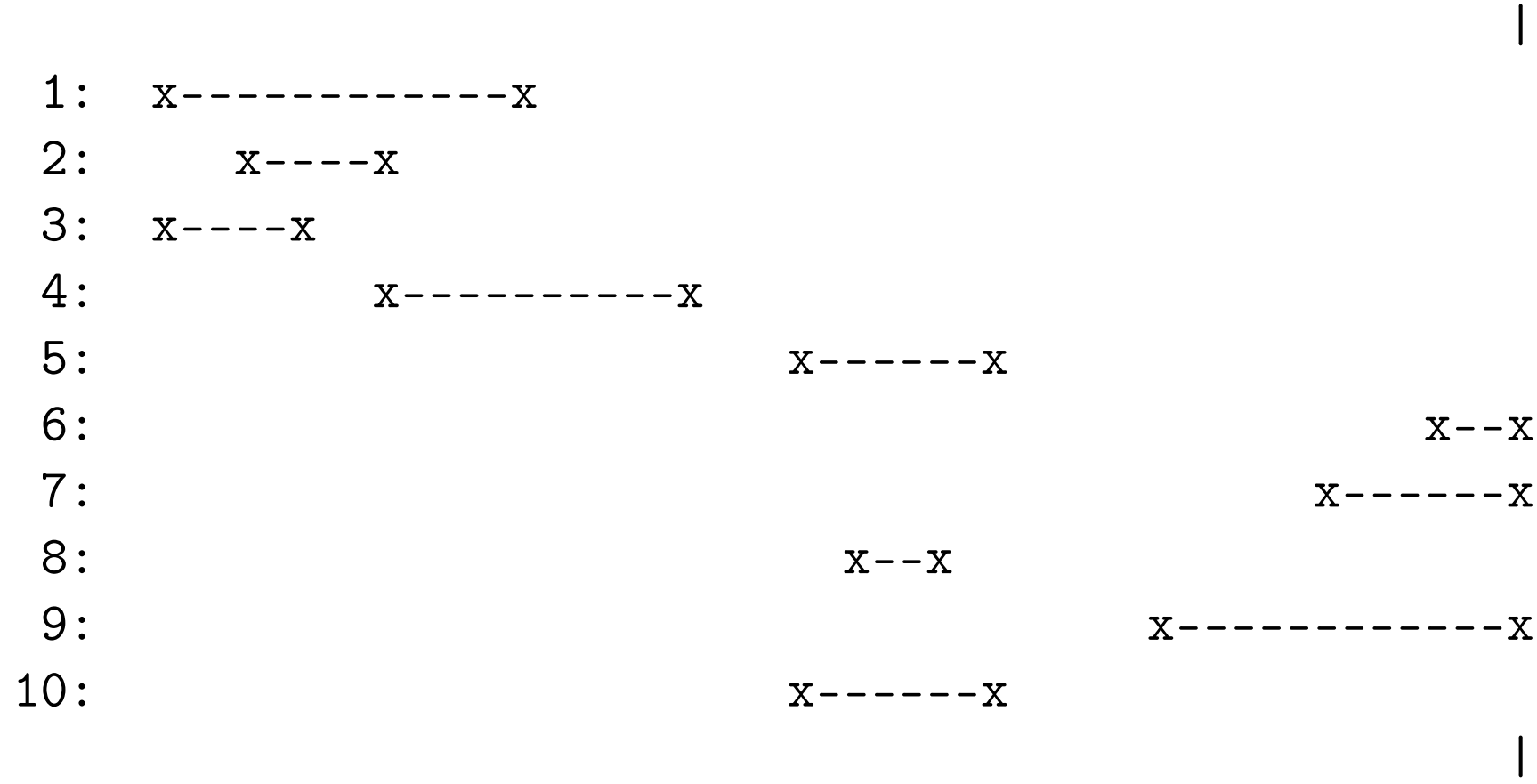
[6, 7, 9]
r = 28



[6, 7, 9]
r = 28

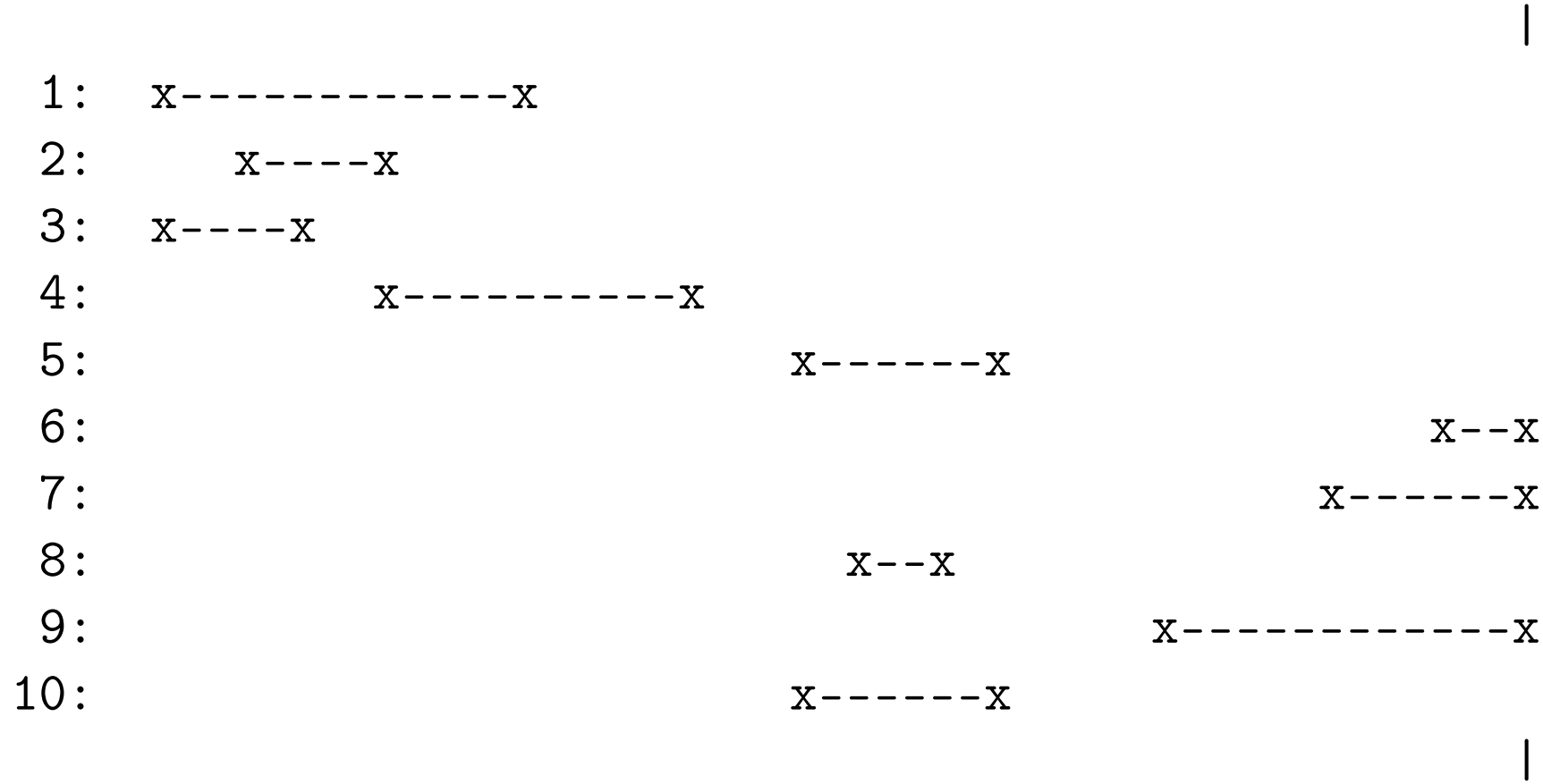


[7, 9]
r = 28



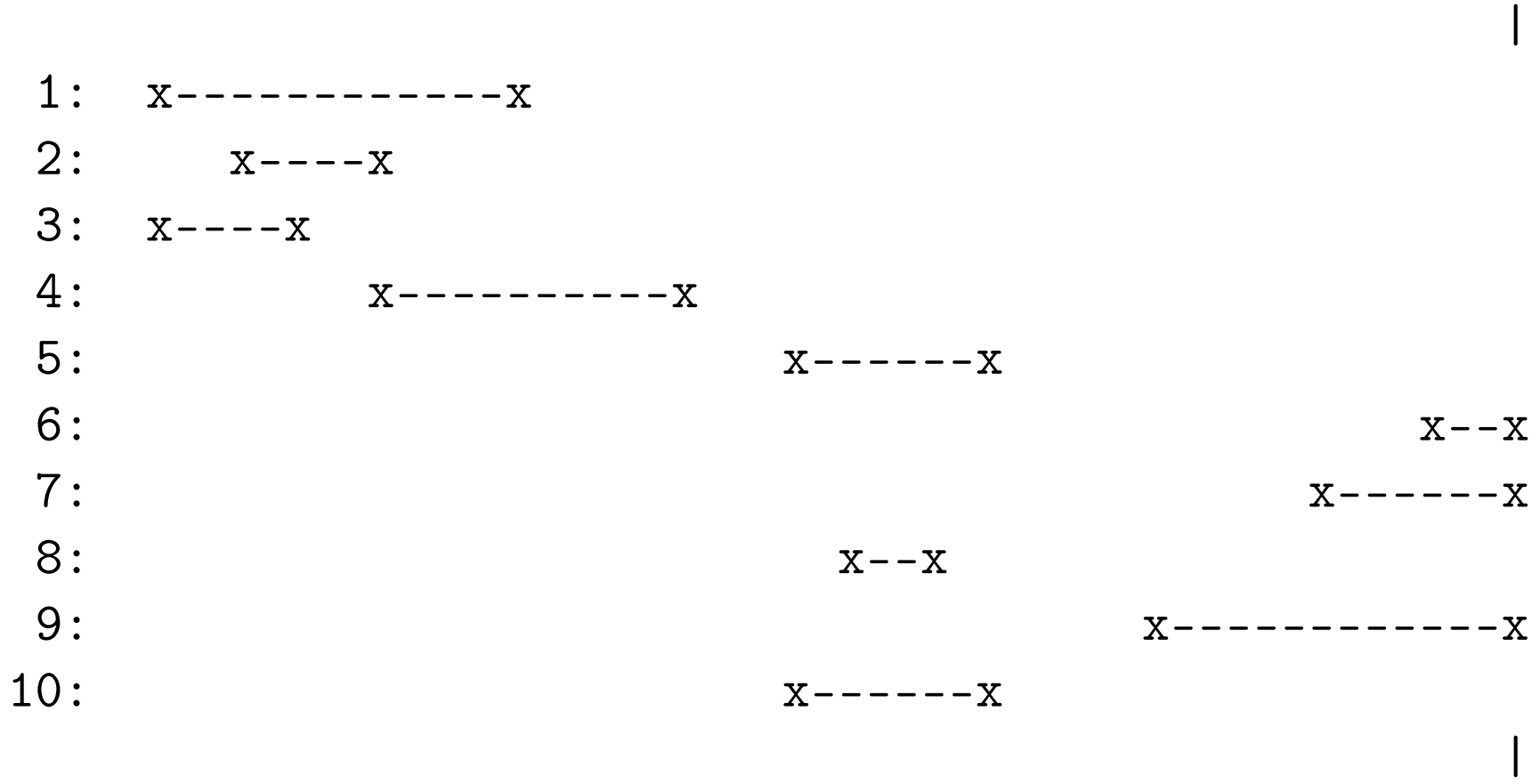
[9]

r = 28



[]

r = 28




```
[]
r = 42
```

```c
#include <stdlib.h>
#include <stdio.h>
#define rep(E, F) for (E = 0; E < (F); E++)
typedef struct { int x, y; } ii;
int cmp(const void* p1, const void* p2) { return ((ii*)p1)->x - ((ii*)p2)->x; }

int main()
{
    int n, r, i, j, k;
    scanf("%d", &n);
    ii a[2*n]; int b[n];
    rep(i, n)
    {
        scanf("%d%d", &(a[2*i].x), &(a[2*i + 1].x));
        a[2*i].y = i; a[2*i + 1].y = i; b[i] = 0;
    }
    qsort(a, 2*n, sizeof(a[0]), cmp);
    i = 0, r = 0;
    while (i < 2*n)
    {
        k = 1, j = i + 1, b[a[i].y] = 1;
        while (k > 0)
        {
            if (b[a[j].y] == 1) k--;
            else b[a[j].y] = 1, k++;
            j++;
        }
        r = r + a[j - 1].x - a[i].x; i = j;
    }
    printf("%d\n", r);
    return 0;
}
```

- 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 $\mathcal{O}(n \log n)$.

Lengsta vaxandi hlutruna (LIS)

- Hlutruna í talnarunu er runa af tölum, allar úr upprunalegu rununni, sem eru í sömu röð og í upprunalegu rununni.
- Hvernig getum við fundið *lengtsu vaxandi hlutrunu* (e. *longest increasing subsequence* (LIS)) í gefinni runu?
- Sem dæmi er `[2 3 5 9]` ein ef lengstu vaxandi hlutrunum `[2 3 1 5 9 8 7]`.

- Gerum ráð fyrir að við höfum talnarunu af lengd $1 \leq n \leq 15$.
- Við getum þá prófað allar hlutrunur, skoðað hvort þær séu vaxandi og geymt þá lengstu.

```c
#include <stdio.h>
#define rep(E, F) for (E = 0; E < (F); E++)

int main()
{
    int n, i, j;
    scanf("%d", &n);
    int a[n];
    rep(i, n) scanf("%d", &a[i]);
    int mx = 0, mxi;
    rep(i, 1 << n)
    {
        int s[n], sc = 0;
        rep(j, n) if (((1 << j)&i) != 0) s[sc++] = a[j];
        rep(j, sc - 1) if (s[j + 1] < s[j]) break;
        if (j == sc - 1 && sc > mx) mx = sc, mxi = i;
    }
    printf("%d\n", mx);
    rep(i, n) if (((1 << i)&mxi) != 0) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

- Við þurfum að skoða öll hlutmengi vísamengis rununnar og fyrir hvert þeirra þarf að ítra í gegnum allt vísamengið.
- Þar sem vísamengið inniheldur n stök hefur þessi lausn tímaflækjuna $\mathcal{O}(n \cdot 2^n)$.

- Það má þó gera þetta hraðar.
- Við getum notað kvika bestun!
- Látum a vera runu af n tölum.
- Látum $f(k)$ vera lengd lengstu hlutrunu sem endar í k-ta staki a.
- Við höfum þá að

$$f(x) = \begin{cases} 1, & \text{ef } x = 1 \\ \max\limits_{\substack{1 \le k < x \\ a_k \le a_x}} f(k) + 1 & \text{annars.} \end{cases}$$

```
 6 int a[MAXN], d[MAXN], n;
 7 int dp_lookup(int x)
 8 {
 9     int i;
10     if (d[x] != -1) return d[x];
11     d[x] = 1;
12     rep(i, x) if (a[i] <= a[x])
13         d[x] = max(d[x], 1 + dp_lookup(i));
14     return d[x];
15 }
```

- Við höfum n stöður og hverja stöðu má reikna í $\mathcal{O}(\, n \,)$ tíma.
- Svo þessi lausn hefur tímaflækjuna $\mathcal{O}(n^2)$ tíma.

- En er hægt að gera þetta ennþá hraðar?
- Heldur betur!
- Við getum notað helmingunarleit.
- Skoðum first reiknirit sem er ekki hentugt að útfæra.

- Höfum lista af listum.
- Skilgreinum röðun þannig að listar eru bornir saman eftir síðasta staki.
- Listalistinn okkar byrjar tómur.
- Löbbum í gegnum `a` í réttri röð og fyrir hvert stak `a[i]` finnum við þann lista sem hefur stærsta aftasta stakið sem er minna en `a[i]`.
- Við afritum nú listann sem við fundum, setjum hann fyrir aftan, bætum stakinu okkar við hann og fjarlægjum listann fyrir aftann nýja listann (ef það er einhver).
- Að þessu loknu er aftasti listinn í listalistanum einn af lengstu vaxandi hlutrununum.
- Rúllum í gegnum þetta fyrir listann `[0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15]`.

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
 |

   []
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
 |

-> []
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
 |

   []
   []
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
 |

   []
   [0]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
    |

   []
   [0]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
    |

   []
-> [0]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
    |

   []
   [0]
   [0]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
    |

   []
   [0]
   [0, 8]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
       |

   []
   [0]
   [0, 8]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
       |

   []
-> [0]
   [0, 8]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
       |

   []
   [0]
   [0]
   [0, 8]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
       |

   []
   [0]
   [0, 4]
   [0, 8]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
       |

   []
   [0]
   [0, 4]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
           |

   []
   [0]
   [0, 4]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
            |

   []
   [0]
-> [0, 4]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
           |

   []
   [0]
   [0, 4]
   [0, 4]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
           |

   []
   [0]
   [0, 4]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                |

   []
   [0]
   [0, 4]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                |

   []
-> [0]
   [0, 4]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                |

   []
   [0]
   [0]
   [0, 4]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                |

   []
   [0]
   [0, 2]
   [0, 4]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                |

   []
   [0]
   [0, 2]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                      |

   []
   [0]
   [0, 2]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                     |

   []
   [0]
-> [0, 2]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                      |

   []
   [0]
   [0, 2]
   [0, 2]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                     |

   []
   [0]
   [0, 2]
   [0, 2, 10]
   [0, 4, 12]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                    |

   []
   [0]
   [0, 2]
   [0, 2, 10]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                         |

   []
   [0]
   [0, 2]
   [0, 2, 10]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                         |

   []
   [0]
-> [0, 2]
   [0, 2, 10]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                          |

   []
   [0]
   [0, 2]
   [0, 2]
   [0, 2, 10]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                          |

   []
   [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 10]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                        |

   []
   [0]
   [0, 2]
   [0, 2, 6]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 2]
   [0, 2, 6]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 2]
-> [0, 2, 6]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
-> [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 1]
   [0, 2]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                            |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                 |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                 |

   []
   [0]
   [0, 1]
-> [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 9]
   [0, 2, 6, 14]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                          |

   []
   [0]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                          |

   []
   [0]
-> [0, 1]
   [0, 2, 6]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                 |

   []
   [0]
   [0, 1]
   [0, 1]
   [0, 2, 6]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                |

   []
   [0]
   [0, 1]
   [0, 1, 5]
   [0, 2, 6]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                |

   []
   [0]
   [0, 1]
   [0, 1, 5]
   [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                 |

   []
   [0]
   [0, 1]
   [0, 1, 5]
-> [0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                   |
```

```
[]
[0]
[0, 1]
[0, 2, 6, 9]
[0, 2, 6, 9]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                  |
```

```
[]
[0]
[0, 1]
[0, 1, 5]
[0, 2, 6, 9]
[0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                  |

   []
   [0]
   [0, 1]
   [0, 1, 5]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                  |


   []
   [0]
-> [0, 1]
   [0, 1, 5]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                   |


   []
   [0]
   [0, 1]
   [0, 1]
   [0, 1, 5]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                  |
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 1, 5]
[0, 2, 6, 9]
[0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                   |


   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
-> [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 2, 6, 9]
[0, 2, 6, 9, 11]
[0, 2, 6, 9, 13]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
-> [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |

   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 1, 3]
   [0, 2, 6, 9]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 1, 3, 7]
[0, 2, 6, 9]
[0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                    |
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 1, 3, 7]
[0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                      |


   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 1, 3, 7]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                      |


   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 1, 3, 7]
-> [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                      |


   []
   [0]
   [0, 1]
   [0, 1, 3]
   [0, 1, 3, 7]
   [0, 2, 6, 9, 11]
   [0, 2, 6, 9, 11]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
                                                      |
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 1, 3, 7]
[0, 2, 6, 9, 11]
[0, 2, 6, 9, 11, 15]
```

```
[0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
```

```
[]
[0]
[0, 1]
[0, 1, 3]
[0, 1, 3, 7]
[0, 2, 6, 9, 11]
[0, 2, 6, 9, 11, 15]
```

- Hvernig útfærum við þetta?
- Við nýtum okkur það að listarnir sem við vorum með eru að mestu óþarfir því við þurfum bara aftasta stakið í hverjum þeirra.
- Við erum þá með lista af tölum, sem gerir allt mun auðveldara.

```
12 int lis(int* a, int n)
13 { // Skilar lengd lengstu hlutruna rununnar |a|.
14     int i, j;
15     int b[n + 2];
16     rep(i, n + 2) b[i] = INF;
17     b[0] = -INF;
18     rep(i, n) b[bs(b, n + 1, a[i])] = a[i];
19     for (i = 0; b[i] != INF; i++);
20     return i - 1;
21 }
```

- Takið eftir að í hverju skrefi notum við helmingunarleit.
- Svo tímaflækjan er $\mathcal{O}(n \log n)$.

Næsta stærra stak (NGE)

- Látum a vera lista af n tölum.
- Við segjum að *næsta stak stærra en a[i]* (e. *next greater element* (NGE)) sé stakið a[j] þ.a. $i < j \leq n$ og a[i] sé minna en a[j], með j valið sem minnst.
- Sem dæmi er NGE miðju stakins 4 í listanum [2, 3, 4, 8, 5] talan 8.
- Til þæginda segjum við að NGE tölunnar 8 í listanum [2, 3, 4, 8, 5] sé -1.
- Það er auðséð að við getum reiknað NGE allra talnanna með tvöfaldri for-lykkju.

```
 4 void nge(int* a, int* b, int n)
 5 {
 6     int i, j;
 7     rep(i, n)
 8     {
 9         rep(j, n - i) if (a[i] < a[i + j]) break;
10         b[i] = (j == n - i ? -1 : i + j);
11     }
12 }
```

- Þar sem þessi lausnir er tvöföld `for`-lykkja, hvor af lengd n, þá er lausnin $\mathcal{O}(n^2)$.

- En þetta má bæta.
- Gefum okkur hlaða `h`.
- Löbbum í gegnum `a` í réttri röð.
- Tökum nú tölur úr hlaðan og setjum `NGE` þeirra talna sem `a[i]` á meðan `a[i]` er stærri en toppurinn á hlaðanum.
- Þegar toppurinn á hlaðanum er stærri en `a[i]` þá látum við `a[i]` á hlaðann og höldum svo áfram.
- Bersýnilega er hlaðinn ávallt raðaður, svo þú færð allar tölur sem eiga að hafa `a[i]` sem `NGE`.
- Þegar búið er að fara í gegnum `a` látum við `NGE` þeirra staka sem eftir eru í `h` vera -1.

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
|

 0 1 2 3 4 5 6 7
[x x x x x x x x]


h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
 |

 0 1 2 3 4 5 6 7
[x x x x x x x x]


h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
 |

 0 1 2 3 4 5 6 7
[x x x x x x x x]


h: [0]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   |

 0 1 2 3 4 5 6 7
[x x x x x x x x]


h: [0]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
 ^ |

 0 1 2 3 4 5 6 7
[x x x x x x x x]


h: [0]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
 ^ |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [0]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
     |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   ^ |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
     |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
     |
```

```
 0 1 2 3 4 5 6 7
[1 x x x x x x x]
```

```
h: [1 2]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
      |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1 2]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
     ^ |

 0 1 2 3 4 5 6 7
[1 x x x x x x x]


h: [1 2]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
     ^ |

 0 1 2 3 4 5 6 7
[1 x 3 x x x x x]


h: [1 2]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   ^   |
```

```
 0 1 2 3 4 5 6 7
[1 x 3 x x x x x]
```

```
h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
   ^   |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 x x x x x]
```

```
h: [1]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
      |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 x x x x x]
```

```
h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
      |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 x x x x x]
```

```
h: [3]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
         |

 0 1 2 3 4 5 6 7
[1 3 3 x x x x x]


h: [3]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
      ^ |

 0 1 2 3 4 5 6 7
[1 3 3 x x x x x]


h: [3]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
      ^ |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [3]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
         |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
         |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
          |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
        ^ |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
          |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
          |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4 5]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
              |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]
```

h: [4 5]

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
           ^ |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4 5]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
              |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]
```

h: [4 5]

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
              |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]
```

h: [4 5 6]

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
                |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4 5 6]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
              ^ |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x x x]


h: [4 5 6]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
              ^ |
```

```
 0 1 2 3 4 5 6 7
[1 3 3 4 x x 7 x]
```

h: [4 5 6]

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
            ^   |

 0 1 2 3 4 5 6 7
[1 3 3 4 x x 7 x]


h: [4 5]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
            ^   |

 0 1 2 3 4 5 6 7
[1 3 3 4 x 7 7 x]


h: [4 5]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
        ^       |

 0 1 2 3 4 5 6 7
[1 3 3 4 x 7 7 x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
        ^     |

 0 1 2 3 4 5 6 7
[1 3 3 4 7 7 7 x]


h: [4]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
                |

 0 1 2 3 4 5 6 7
[1 3 3 4 7 7 7 x]


h: []
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
                |

 0 1 2 3 4 5 6 7
[1 3 3 4 7 7 7 x]


h: [8]
```

```
 0 1 2 3 4 5 6 7
[2 3 1 5 7 6 4 8]
```

```
 0 1 2 3 4 5 6 7
[1 3 3 4 7 7 7 x]
```

```
h: [8]
```

```
 4 void nge(int* a, int* b, int n)
 5 { // Eftir kall inniheldur |b| naestu staerstu stok fyrir hvert stak i |a|.
 6     int s[n], c = 0, i;
 7     rep(i, n)
 8     {
 9         while (c > 0 && a[s[c - 1]] < a[i]) b[s[--c]] = i;
10         s[c++] = i;
11     }
12     while (c > 0) b[s[--c]] = -1;
13 }
```

- Við setjum hverja tölu í hlaðann að mestu einu sinni og tökum hana svo úr hlaðanum.
- Svo tímaflækjan er $\mathcal{O}(\, n \,)$.