Contents

1 ubuntu

1.1 run

```
~$ bash cp.sh PA
```

1.2 cp.sh

```
#!/bin/bash
clear
g++ $1.cpp -DDBG -o $1
if [[ "$?" == "0" ]]; then
        echo Running
        ./$1 < $1.in > $1.out
        echo END
fi
```

2 Basic

2.1 ascii

int	char	int	char	int	char
32		64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

2.2 limits

```
[Type]             [size]       [range]
char                 1         127 to -128
signed char          1         127 to -128
unsigned char        1         0 to 255
short                2         32767 to -32768
int                  4         2147483647 to -2147483648
unsigned int         4         0 to 4294967295
long                 4         2147483647 to -2147483648
unsigned long        4         0 to 18446744073709551615
long long            8
           9223372036854775807 to -9223372036854775808
double               8      1.79769e+308 to 2.22507e-308
long double          16    1.18973e+4932 to 3.3621e-4932
float                4        3.40282e+38 to 1.17549e-38
unsigned long long   8         0 to 18446744073709551615
string               32
```

3 字串

3.1 最長迴文子字串

```
#include<bits/stdc++.h>
#define T(x) ((x)%2 ? s[(x)/2] : '.')
using namespace std;

string s;
int n;

int ex(int l,int r){
  int i=0;
```

```
  while(l-i>=0&&r+i<n&&T(l-i)==T(r+i)) i++;
  return i;
}

int main(){
  cin>>s;
  n=2*s.size()+1;
  int mx=0;
  int center=0;
  vector<int> r(n);
  int ans=1;
  r[0]=1;
  for(int i=1;i<n;i++){
    int ii=center-(i-center);
    int len=mx-i+1;
    if(i>mx){
      r[i]=ex(i,i);
      center=i;
      mx=i+r[i]-1;
    }
    else if(r[ii]==len){
      r[i]=len+ex(i-len,i+len);
      center=i;
      mx=i+r[i]-1;
    }
    else r[i]=min(r[ii],len);
    ans=max(ans,r[i]);
  }
  cout<<ans-1<<"\n";
  return 0;
}
```

3.2 stringstream

```
string s,word;
stringstream ss;
getline(cin,s);
ss<<s;
while(ss>>word) cout<<word<<endl;
```

4 STL

4.1 priority_queue

```
priority_queue： 優先隊列，資料預設由大到小排序。

讀取優先權最高的值：
    x = pq.top();
    pq.pop();                   //讀取後刪除
判斷是否為空的priority_queue：
    pq.empty()                  //回傳true
    pq.size()                   //回傳0
如需改變priority_queue的優先權定義：
    priority_queue<T> pq;       //預設由大到小
    priority_queue<T, vector<T>, greater<T> > pq;
                                //改成由小到大
    priority_queue<T, vector<T>, cmp> pq;   //cmp
```

4.2 deque

```
deque 是 C++ 標準模板函式庫
     (Standard Template Library, STL)
    中的雙向佇列容器 (Double-ended Queue) ，
    跟 vector 相似，不過在 vector
        中若是要添加新元素至開端，
    其時間複雜度為 O(N)，但在 deque 中則是 O(1)。
    同樣也能在我們需要儲存更多元素的時候自動擴展空間，
    讓我們不必煩惱佇列長度的問題。
dq.push_back()  //在 deque 的最尾端新增元素
dq.push_front() //在 deque 的開頭新增元素
dq.pop_back()   //移除 deque 最尾端的元素
dq.pop_front()  //移除 deque 最開頭的元素
dq.back()       //取出 deque 最尾端的元素
dq.front()      //回傳 deque 最開頭的元素
dq.insert()
dq.insert(position, n, val)
    position: 插入元素的 index 值
    n: 元素插入次數
    val: 插入的元素值
dq.erase()
    //刪除元素，需要使用迭代器指定刪除的元素或位置，
                //同時也會返回指向刪除元素下一元素的迭代器。
dq.clear()      //清空整個 deque 佇列。
dq.size()       //檢查 deque 的尺寸
dq.empty()      //如果 deque 佇列為空返回 1；
                //若是存在任何元素，則返回0
dq.begin()      //返回一個指向 deque 開頭的迭代器
dq.end()        //指向 deque 結尾，
                //不是最後一個元素，
                //而是最後一個元素的下一個位置
```

4.3 map

```
map：存放 key-value pairs 的映射資料結構，
     會按 key 由小到大排序。
元素存取
operator[]：存取指定的[i]元素的資料

迭代器
begin()：回傳指向map頭部元素的迭代器
end()：回傳指向map末尾的迭代器
rbegin()：回傳一個指向map尾部的反向迭代器
rend()：回傳一個指向map頭部的反向迭代器

遍歷整個map時，利用iterator操作：
取key：it->first 或 (*it).first
取value：it->second 或 (*it).second

容量
empty()：檢查容器是否為空，空則回傳true
size()：回傳元素數量
max_size()：回傳可以容納的最大元素個數

修改器
clear()：刪除所有元素
insert()：插入元素
erase()：刪除一個元素
swap()：交換兩個map

查找
count()：回傳指定元素出現的次數
find()：查找一個元素

//實作範例
#include <bits/stdc++.h>
using namespace std;
int main(){
    //declaration container and iterator
    map<string, string> mp;
    map<string, string>::iterator iter;
    map<string, string>::reverse_iterator iter_r;

    //insert element
    mp.insert(pair<string, string>
            ("r000", "student_zero"));
    mp["r123"] = "student_first";
    mp["r456"] = "student_second";

    //traversal
```

```
    for(iter=mp.begin();iter!=mp.end();iter++)
        cout<<iter->first<<" "
                    <<iter->second<<endl;
    for(iter_r=mp.rbegin();iter_r!=mp.rend();iter_r++)
        cout<<iter_r->first<<"
             "<<iter_r->second<<endl;

    //find and erase the element
    iter=mp.find("r123");
    mp.erase(iter);
    iter=mp.find("r123");
    if(iter!=mp.end())
       cout<<"Find, the value is "
                <<iter->second<<endl;
    else cout<<"Do not Find"<<endl;
    return 0;
}
```

4.4 unordered_map

```
unordered_map：存放 key-value pairs
               的「無序」映射資料結構。
用法與map相同
```

4.5 set

```
set： 集合，去除重複的元素，資料由小到大排序。

取值： 使用iterator
    x = *st.begin();
            // set中的第一個元素(最小的元素)。
    x = *st.rbegin();
            // set中的最後一個元素(最大的元素)。

判斷是否為空的set：
    st.empty() 回傳true
    st.size() 回傳零

常用來搭配的member function：
    st.count(x);
    auto it = st.find(x);
        // binary search, O(log(N))
    auto it = st.lower_bound(x);
        // binary search, O(log(N))
    auto it = st.upper_bound(x);
        // binary search, O(log(N))
```

4.6 multiset

```
與 set 用法雷同，但會保留重複的元素。
資料由小到大排序。
宣告：
    multiset<int> st;
刪除資料：
    st.erase(val);
    //會刪除所有值為 val 的元素。
    st.erase(st.find(val));
    //只刪除第一個值為 val 的元素。
```

4.7 unordered_set

```
unordered_set 的實作方式通常是用雜湊表(hash table)，
資料插入和查詢的時間複雜度很低，為常數級別O(1)，
相對的代價是消耗較多的記憶體，空間複雜度較高，
無自動排序功能。

unordered_set 判斷元素是否存在
```

```
unordered_set<int> myunordered_set;
myunordered_set.insert(2);
myunordered_set.insert(4);
myunordered_set.insert(6);
cout << myunordered_set.count(4) << "\n"; // 1
cout << myunordered_set.count(8) << "\n"; // 0
```

4.8 單調隊列

```
//單調隊列
"如果一個選手比你小還比你強，你就可以退役了。"--單調隊列

example

給出一個長度為 n 的數組，
輸出每 k 個連續的數中的最大值和最小值。

#include <bits/stdc++.h>
#define maxn 1000100
using namespace std;
int q[maxn], a[maxn];
int n, k;

void getmin() {
    // 得到這個隊列裡的最小值，直接找到最後的就行了
    int head=0,tail=0;
    for(int i=1;i<k;i++) {
        while(head<=tail&&a[q[tail]]>=a[i]) tail--;
        q[++tail]=i;
    }
    for(int i=k; i<=n;i++) {
        while(head<=tail&&a[q[tail]]>=a[i]) tail--;
        q[++tail]=i;
        while(q[head]<=i-k) head++;
        cout<<a[q[head]]<<" ";
    }
    cout<<endl;
}

void getmax() {  // 和上面同理
    int head=0,tail=0;
    for(int i=1;i<k;i++) {
        while(head<=tail&&a[q[tail]]<=a[i])tail--;
        q[++tail]=i;
    }
    for(int i=k;i<=n;i++) {
        while(head<=tail&&a[q[tail]]<=a[i])tail--;
        q[++tail]=i;
        while(q[head]<=i-k) head++;
        cout<<a[q[head]]<<" ";
    }
    cout<<endl;
}

int main(){
    cin>>n>>k;  //每k個連續的數
    for(int i=1;i<=n;i++) cin>>a[i];
    getmin();
    getmax();
    return 0;
}
```

5 sort

5.1 大數排序

```
#python大數排序

while True:
  try:
    n = int(input())           # 有幾筆數字需要排序
```

```
    arr = []                    # 建立空串列
    for i in range(n):
      arr.append(int(input())) # 依序將數字存入串列
    arr.sort()                  # 串列排序
    for i in arr:
      print(i)                # 依序印出串列中每個項目
  except:
    break
```

6 math

6.1 質數與因數

```
埃氏篩法
int n;
vector<int> isprime(n+1,1);
isprime[0]=isprime[1]=0;
for(int i=2;i*i<=n;i++){
    if(isprime[i])
        for(int j=i*i;j<=n;j+=i) isprime[j]=0;
}

歐拉篩O(n)
#define MAXN 47000 //sqrt(2^31)=46,340...
bool isPrime[MAXN];
int prime[MAXN];
int primeSize=0;
void getPrimes(){
    memset(isPrime,true,sizeof(isPrime));
    isPrime[0]=isPrime[1]=false;
    for(int i=2;i<MAXN;i++){
        if(isPrime[i]) prime[primeSize++]=i;
        for(int
            j=0;j<primeSize&&i*prime[j]<=MAXN;++j){
            isPrime[i*prime[j]]=false;
            if(i%prime[j]==0) break;
        }
    }
}

最大公因數 O(log(min(a,b)))
int GCD(int a,int b){
    if(b==0) return a;
    return GCD(b,a%b);
}

質因數分解
void primeFactorization(int n){
    for(int i=0;i<(int)p.size();++i){
        if(p[i]*p[i]>n) break;
        if(n%p[i]) continue;
        cout<<p[i]<<' ';
        while(n%p[i]==0) n/=p[i];
    }
    if(n!=1) cout<<n<<' ';
    cout<<'\n';
}

擴展歐幾里得算法
//ax+by=GCD(a,b)
#include <bits/stdc++.h>
using namespace std;

int ext_euc(int a,int b,int &x,int &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    int d=ext_euc(b,a%b,y,x);
    y-=a/b*x;
    return d;
}

int main(){
    int a,b,x,y;
    cin>>a>>b;
    ext_euc(a,b,x,y);
    cout<<x<<' '<<y<<endl;
    return 0;
}



歌德巴赫猜想
solution : 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
#include <iostream>
using namespace std;
#define N 20000000
int ox[N],p[N],pr;
void PrimeTable(){
    ox[0]=ox[1]=1;
    pr=0;
    for(int i=2;i<N;i++){
        if(!ox[i]) p[pr++]=i;
        for(int j=0;i*p[j]<N&&j<pr;j++)
            ox[i*p[j]]=1;
    }
}

int main(){
    PrimeTable();
    int n;
    while(cin>>n,n){
        int x;
        for(x=1;;x+=2)
            if(!ox[x]&&!ox[n-x]) break;
        printf("%d = %d + %d\n",n,x,n-x);
    }
}
problem : 給定整數 N，
        求 N 最少可以拆成多少個質數的和。
如果 N 是質數，則答案為 1。
如果 N 是偶數(不包含2)，則答案為 2 (強歌德巴赫猜想)。
如果 N 是奇數且 N-2 是質數，則答案為 2 (2+質數)。
其他狀況答案為 3 (弱歌德巴赫猜想)。
#include<bits/stdc++.h>
using namespace std;

bool isPrime(int n){
    for(int i=2;i<n;++i){
        if(i*i>n) return true;
        if(n%i==0) return false;
    }
    return true;
}

int main(){
    int n;
    cin>>n;
    if(isPrime(n)) cout<<"1\n";
    else if(n%2==0||isPrime(n-2)) cout<<"2\n";
    else cout<<"3\n";
}
```

6.2 快速冪

```
計算a^b
#include<iostream>
#define ll long long
using namespace std;

const ll MOD=1000000007;
ll fp(ll a, ll b) {
    int ans=1;
    while(b>0){
        if(b&1) ans=ans*a%MOD;
        a=a*a%MOD;
        b>>=1;
```

```
    }
    return ans;
}

int main() {
  int a,b;
  cin>>a>>b;
  cout<<fp(a,b);
}
```

6.3 歐拉函數

```
//計算閉區間 [1,n] 中的正整數與 n 互質的個數

int phi(){
    int ans=n;
    for(int i=2;i*i<=n;i++)
        if(n%i==0){
            ans=ans-ans/i;
            while(n%i==0) n/=i;
        }
    if(n>1) ans=ans-ans/n;
    return ans;
}
```

6.4 atan

```
說明
  atan() 和 atan2() 函數分別計算 x 和 y/x的反正切。

回覆值
  atan() 函數會傳回介於範圍 - /2 到  /2 弧度之間的值。
  atan2() 函數會傳回介於 -  至   弧度之間的值。
  如果 atan2() 函數的兩個引數都是零，
  則函數會將 errno 設為 EDOM，並傳回值 0。

範例
#include <math.h>
#include <stdio.h>

int main(void){
    double a,b,c,d;

    c=0.45;
    d=0.23;

    a=atan(c);
    b=atan2(c,d);

    printf("atan(%lf)=%lf/n",c,a);
    printf("atan2(%lf,%lf)=%lf/n",c,d,b);

}

/*
atan(0.450000)=0.422854
atan2(0.450000,0.230000)=1.098299
*/
```

6.5 大步小步

```
  題意
給定 B,N,P，求出 L 滿足 B^L N(mod P)。

  題解
餘數的循環節長度必定為 P 的因數，因此
    B^0 B^P,B^1 B^(P+1),…，
也就是說如果有解則 L<N，枚舉0,1,2,L-1
    能得到結果，但會超時。

將 L 拆成 mx+y，只要分別枚舉 x,y 就能得到答案，
設 m=√P 能保證最多枚舉 2√P 次 。

B^(mx+y) N(mod P)
B^(mx)B^y N(mod P)
B^y N(B^(-m))^x (mod P)

先求出 B^0,B^1,B^2,...,B^(m-1)，
再枚舉 N(B^(-m)),N(B^(-m))^2,… 查看是否有對應的 B^y。
這種算法稱為大步小步演算法，
大步指的是枚舉 x (一次跨 m 步)，
小步指的是枚舉 y (一次跨 1 步)。

  複雜度分析
利用 map/unorder_map 存放 B^0,B^1,B^2,…,B^(m-1)，
枚舉 x 查詢 map/unorder_map 是否有對應的 B^y，
存放和查詢最多 2√P 次，時間複雜度為 O(√Plog√P)/O(√P)。



#include <bits/stdc++.h>
using namespace std;
using LL = long long;
LL B, N, P;

LL fpow(LL a,LL b,LL c){
    LL res=1;
    for(;b;b >>=1){
        if(b&1)
            res=(res*a)%c;
        a=(a*a)%c;
    }
    return res;
}

LL BSGS(LL a,LL b,LL p){
    a%=p,b%=p;
    if(a==0)
        return b==0?1:-1;
    if(b==1)
        return 0;
    map<LL, LL> tb;
    LL sq=ceil(sqrt(p-1));
    LL inv=fpow(a,p-sq-1,p);
    tb[1]=sq;
    for(LL i=1,tmp=1;i<sq;++i){
        tmp=(tmp*a)%p;
        if(!tb.count(tmp))
            tb[tmp]=i;
    }
    for(LL i=0;i<sq;++i){
        if(tb.count(b)){
            LL res=tb[b];
            return i*sq+(res==sq?0:res);
        }
        b=(b*inv)%p;
    }
    return -1;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    while(cin>>P>>B>>N){
        LL ans=BSGS(B,N,P);
        if(ans==-1)
            cout<<"no solution\n";
        else
            cout<<ans<<'\n';
    }
}
```

7 algorithm

7.1 basic

```
min_element：找尋最小元素
min_element(first, last)
max_element：找尋最大元素
max_element(first, last)
sort：排序，預設由小排到大。
sort(first, last)
sort(first, last, cmp)：可自行定義比較運算子 cmp 。
find：尋找元素。
find(first, last, val)
lower_bound：尋找第一個小於 x 的元素位置，
             如果不存在，則回傳 last 。
lower_bound(first, last, val)
upper_bound：尋找第一個大於 x 的元素位置，
             如果不存在，則回傳 last 。
upper_bound(first, last, val)
next_permutation：將序列順序轉換成下一個字典序，
                  如果存在回傳 true，反之回傳 false。
next_permutation(first, last)
prev_permutation：將序列順序轉換成上一個字典序，
                  如果存在回傳 true，反之回傳 false。
prev_permutation(first, last)
```

7.2 二分搜

```cpp
int binary_search(int target) {
// For range [ok, ng) or (ng, ok], "ok" is for the
// index that target value exists, with "ng" doesn't.
    int ok = maxn, ng = -1;
// For first lower_bound, ok=maxn and ng=-1,
// for last lower_bound, ok = -1 and ng = maxn
// (the "check" funtion
// should be changed depending on it.)
    while(abs(ok - ng) > 1) {
        int mid = (ok + ng) >> 1;
        if(check(mid)) ok = mid;
        else ng = mid;
// Be careful, "arr[mid]>=target" for first
// lower_bound and "arr[mid]<=target" for
// last lower_bound. For range (ng, ok],
// convert it into (ng, mid] and (mid, ok] than
// choose the first one, or convert [ok, ng) into
// [ok, mid) and [mid, ng) and than choose
// the second one.
    }
    return ok;
}

lower_bound(arr, arr + n, k);    //最左邊 ≥ k 的位置
upper_bound(arr, arr + n, k);    //最左邊 > k 的位置
upper_bound(arr, arr + n, k) - 1;//最右邊 ≤ k 的位置
lower_bound(arr, arr + n, k) - 1;//最右邊 < k 的位置
(lower_bound, upper_bound)       //等於 k 的範圍
equal_range(arr, arr+n, k);
```

7.3 三分搜

```
  題意
給定兩射線方向和速度，問兩射線最近距離。

  題解
假設 F(t) 為兩射線在時間 t 的距離，F(t) 為二次函數，
可用三分搜找二次函數最小值。
```

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Point{
    double x, y, z;
    Point() {}
    Point(double _x,double _y,double _z):
        x(_x),y(_y),z(_z){}
    void read() { cin>>x>>y>>z; }
    Point operator+(const Point &rhs) const{
        return Point(x+rhs.x,y+rhs.y,z+rhs.z);
    }
    Point operator-(const Point &rhs) const{
        return Point(x-rhs.x,y-rhs.y,z-rhs.z);
    }
    Point operator*(const double &d) const{
        return Point(x*d,y*d,z*d);
    }
    Point operator/(const double &d) const{
        return Point(x/d,y/d,z/d);
    }
    double dist(const Point &rhs) const{
        double res = 0;
        res+=(x-rhs.x)*(x-rhs.x);
        res+=(y-rhs.y)*(y-rhs.y);
        res+=(z-rhs.z)*(z-rhs.z);
        return res;
    }
};

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    int T;
    cin>>T;
    for(int ti=1;ti<=T;++ti){
        double time;
        Point x1,y1,d1,x2,y2,d2;
        cin>>time;
        x1.read();
        y1.read();
        x2.read();
        y2.read();
        d1=(y1-x1)/time;
        d2=(y2-x2)/time;
        double L=0,R=1e8,m1,m2,f1,f2;
        double ans = x1.dist(x2);
        while(abs(L-R)>1e-10){
            m1=(L+R)/2;
            m2=(m1+R)/2;
            f1=((d1*m1)+x1).dist((d2*m1)+x2);
            f2=((d1*m2)+x1).dist((d2*m2)+x2);
            ans = min(ans,min(f1,f2));
            if(f1<f2) R=m2;
            else L=m1;
        }
        cout<<"Case "<<ti<<": ";
        cout<<fixed<<setprecision(4)<<sqrt(ans)<<'\n';
    }
}
```

7.4 prefix sum

```
// 前綴和
陣列前n項的和。
b[i]=a[0]+a[1]+a[2]+ … +a[i]
區間和 [l, r]：b[r]-b[l-1] (要保留b[l]所以-1)
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin>>n;
    int a[n],b[n];
    for(int i=0;i<n;i++) cin>>a[i];
    b[0]=a[0];
    for(int i=1;i<n;i++) b[i]=b[i-1]+a[i];
```

```
    for(int i=0;i<n;i++) cout<<b[i]<<' ';
    cout<<'\n';
    int l,r;
    cin>>l>>r;
    cout<<b[r]-b[l-1]; //區間和
}
```

7.5 差分

```
// 差分
用途：在區間 [l, r] 加上一個數字v。
b[l] += v; (b[0~l] 加上v)
b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v) )
給的 a[] 是前綴和數列，建構 b[]，
因為 a[i] = b[0] + b[1] + b[2] + ⋯ + b[i]，
所以 b[i] = a[i] − a[i-1]。
在 b[l] 加上 v，b[r+1] 減去 v，
最後再從 0 跑到 n 使 b[i] += b[i-1]。
這樣一來，b[] 是一個在某區間加上v的前綴和。

#include <bits/stdc++.h>
using namespace std;
int a[1000], b[1000];
// a: 前綴和數列, b: 差分數列
int main(){
    int n, l, r, v;
    cin >> n;
    for(int i=1; i<=n; i++){
        cin >> a[i];
        b[i] = a[i] - a[i-1]; //建構差分數列
    }
    cin >> l >> r >> v;
    b[l] += v;
    b[r+1] -= v;

    for(int i=1; i<=n; i++){
        b[i] += b[i-1];
        cout << b[i] << ' ';
    }
}
```

7.6 greedy

```
//貪心
貪心演算法的核心為，
採取在目前狀態下最好或最佳（即最有利）的選擇。
貪心演算法雖然能獲得當前最佳解，
但不保證能獲得最後（全域）最佳解，
提出想法後可以先試圖尋找有沒有能推翻原本的想法的反例，
確認無誤再實作。


刪數字問題
//problem
給定一個數字 N(≤10^100)，需要刪除 K 個數字，
請問刪除 K 個數字後最小的數字為何?

//solution
刪除滿足第 i 位數大於第 i+1 位數的最左邊第 i 位數，
扣除高位數的影響較扣除低位數的大。

//code
int main(){
    string s;
    int k;
    cin>>s>>k;
    for(int i=0;i<k;++i){
        if((int)s.size()==0) break;
        int pos =(int)s.size()-1;
        for(int j=0;j<(int)s.size()-1;++j){
            if(s[j]>s[j+1]){
                pos=j;
                break;
            }
        }
        s.erase(pos,1);
    }
    while((int)s.size()>0&&s[0]=='0')
        s.erase(0,1);
    if((int)s.size()) cout<<s<<'\n';
    else cout<<0<<'\n';
}


最小區間覆蓋長度
//problem
給定 n 條線段區間為 [Li,Ri]，
請問最少要選幾個區間才能完全覆蓋 [0,S]?

//solution
先將所有區間依照左界由小到大排序，
對於當前區間 [Li,Ri]，要從左界 >Ri 的所有區間中，
找到有著最大的右界的區間，連接當前區間。

//problem
長度 n 的直線中有數個加熱器，
在 x 的加熱器可以讓 [x-r,x+r] 內的物品加熱，
問最少要幾個加熱器可以把 [0,n] 的範圍加熱。

//solution
對於最左邊沒加熱的點a，選擇最遠可以加熱a的加熱器，
更新已加熱範圍，重複上述動作繼續尋找加熱器。

//code
int main(){
    int n, r;
    int a[1005];
    cin>>n>>r;
    for(int i=1;i<=n;++i) cin>>a[i];
    int i=1,ans=0;
    while(i<=n){
        int R=min(i+r-1,n),L=max(i-r+1,0)
        int nextR=-1;
        for(int j=R;j>=L;--j){
            if(a[j]){
                nextR=j;
                break;
            }
        }
        if(nextR==-1){
            ans=-1;
            break;
        }
        ++ans;
        i=nextR+r;
    }
    cout<<ans<<'\n';
}


最多不重疊區間
//problem
給你 n 條線段區間為 [Li,Ri]，
請問最多可以選擇幾條不重疊的線段(頭尾可相連)?

//solution
依照右界由小到大排序，
每次取到一個不重疊的線段，答案 +1。

//code
struct Line{
    int L,R;
    bool operator<(const Line &rhs)const{
        return R<rhs.R;
    }
};
```

```cpp
int main(){
    int t;
    cin>>t;
    Line a[30];
    while(t--){
        int n=0;
        while(cin>>a[n].L>>a[n].R,a[n].L||a[n].R)
            ++n;
        sort(a,a+n);
        int ans=1,R=a[0].R;
        for(int i=1;i<n;i++){
            if(a[i].L>=R){
                ++ans;
                R=a[i].R;
            }
        }
        cout<<ans<<'\n';
    }
}
```

最小化最大延遲問題

```cpp
//problem
```

給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，第 i 項工作延遲的時間為 Li=max(0,Fi-Di)，
原本Fi 為第 i 項工作的完成時間，
求一種工作排序使 maxLi 最小。

```cpp
//solution
```

按照到期時間從早到晚處理。

```cpp
//code
struct Work{
    int t, d;
    bool operator<(const Work &rhs)const{
        return d<rhs.d;
        }
};

int main(){
    int n;
    Work a[10000];
    cin>>n;
    for(int i=0;i<n;++i)
        cin>>a[i].t>>a[i].d;
    sort(a,a+n);
    int maxL=0,sumT=0;
    for(int i=0;i<n;++i){
        sumT+=a[i].t;
        maxL=max(maxL,sumT-a[i].d);
    }
    cout<<maxL<<'\n';
}
```

最少延遲數量問題

```cpp
//problem
```

給定 N 個工作，每個工作的需要處理時長為 Ti，
期限是 Di，求一種工作排序使得逾期工作數量最小。

```cpp
//solution
```

期限越早到期的工作越先做。將工作依照到期時間從早到晚排序，
依序放入工作列表中，如果發現有工作預期，
就從目前選擇的工作中，移除耗時最長的工作。

上述方法為 Moore-Hodgson s Algorithm。

```cpp
//problem
```

給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜?

```cpp
//solution
```

和最少延遲數量問題是相同的問題，只要將題敘做轉換。
工作處裡時長 → 烏龜重量
工作期限 → 烏龜可承受重量
多少工作不延期 → 可以疊幾隻烏龜

```cpp

//code
struct Work{
    int t, d;
    bool operator<(const Work &rhs)const{
        return d<rhs.d;
        }
};

int main(){
    int n=0;
    Work a[10000];
    priority_queue<int> pq;
    while(cin>>a[n].t>>a[n].d)
        ++n;
    sort(a,a+n);
    int sumT=0,ans=n;
    for(int i=0;i<n;++i){
        pq.push(a[i].t);
        sumT+=a[i].t;
        if(a[i].d<sumT){
            int x=pq.top();
            pq.pop();
            sumT-=x;
            --ans;
        }
    }
    cout<<ans<<'\n';
}
```

任務調度問題

```cpp
//problem
```

給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，如果第 i 項工作延遲需要受到 pi 單位懲罰，
請問最少會受到多少單位懲罰。

```cpp
//solution
```

依照懲罰由大到小排序，
每項工作依序嘗試可不可以放在 Di-Ti+1,Di-Ti,...,1,0，
如果有空間就放進去，否則延後執行。

```cpp
//problem
```

給定 N 項工作，每項工作的需要處理時長為 Ti，
期限是 Di，如果第 i 項工作在期限內完成會獲得 ai
單位獎勵，
請問最多會獲得多少單位獎勵。

```cpp
//solution
```

和上題相似，這題變成依照獎勵由大到小排序。

```cpp
//code
struct Work{
    int d,p;
    bool operator<(const Work &rhs)const{
        return p>rhs.p;
        }
};

int main(){
    int n;
    Work a[100005];
    bitset<100005> ok;
    while(cin>>n){
        ok.reset();
        for(int i=0;i<n;++i)
            cin>>a[i].d>>a[i].p;
        sort(a,a+n);
        int ans=0;
        for(int i=0;i<n;++i){
            int j=a[i].d;
            while(j--)
                if(!ok[j]){
                    ans+=a[i].p;
                    ok[j]=true;
                    break;
                }
```

```
        }
        cout<<ans<<'\n';
    }
}
```

7.7 floyd warshall

```
int w[n][n];
int d[n][n];
int medium[n][n];
// 由i點到j點的路徑，其中繼點為medium[i][j]。

void floyd_warshall(){ //O(V^3)
  for(int i=0;i<n;i++)
    for(int j=0;j<n;j++){
      d[i][j]=w[i][j];
      medium[i][j]=-1;
      // 預設為沒有中繼點
    }
  for(int i=0;i<n;i++) d[i][i]=0;
  for(int k=0;k<n;k++)
    for(int i=0;i<n;i++)
      for(int j=0;j<n;j++)
        if(d[i][k]+d[k][j]<d[i][j]){
          d[i][j]=d[i][k]+d[k][j];
          medium[i][j]=k;
          // 由i點走到j點經過了k點
        }
}

// 這支函式並不會印出起點和終點，必須另行印出。
void find_path(int s,int t){     // 印出最短路徑
  if(medium[s][t]==-1) return;   // 沒有中繼點就結束
  find_path(s,medium[s][t]);     // 前半段最短路徑
  cout<<medium[s][t];      // 中繼點
  find_path(medium[s][t],t);     // 後半段最短路徑
}
```

7.8 dinic

```
#include <stdio.h>
#include <string.h>
#include <queue>
#define MAXNODE 105
#define oo 1e9
using namespace std;

int nodeNum;
int graph[MAXNODE][MAXNODE];
int levelGraph[MAXNODE];
bool canReachSink[MAXNODE];

bool bfs(int from, int to){
    memset(levelGraph,0,sizeof(levelGraph));
    levelGraph[from]=1;
    queue<int> q;
    q.push(from);
    int currentNode;
    while(!q.empty()){
        currentNode=q.front();
        q.pop();
        for(int nextNode=1;nextNode<=nodeNum
                                ;++nextNode){
            if((levelGraph[nextNode]==0)&&
                graph[currentNode][nextNode]>0){
                levelGraph[nextNode]=
                    levelGraph[currentNode]+1;
                q.push(nextNode);
            }
            if((nextNode==to)&&
                (graph[currentNode][nextNode]>0))
                return true;
        }
    }
    return false;
}
int dfs(int from, int to, int bottleNeck){
    if(from == to) return bottleNeck;
    int outFlow = 0;
    int flow;
    for(int nextNode=1;nextNode<=nodeNum;++nextNode){
        if((graph[from][nextNode]>0)&&
            (levelGraph[from]==levelGraph[nextNode]-1)&&
            canReachSink[nextNode]){
            flow=dfs(nextNode,to,
                min(graph[from][nextNode],bottleNeck));
            graph[from][nextNode]-=flow; //貪心
            graph[nextNode][from]+=flow; //反悔路
            outFlow+=flow;
            bottleNeck-=flow;
        }
        if(bottleNeck==0) break;
    }
    if(outFlow==0) canReachSink[from]=false;
    return outFlow;
}

int dinic(int from, int to){
    int maxFlow=0;
    while(bfs(from, to)){
        memset(canReachSink,1,sizeof(canReachSink));
        maxFlow += dfs(from, to, oo);
    }
    return maxFlow;
}

int main(){
    int from, to, edgeNum;
    int NetWorkNum = 1;
    int maxFlow;
    while(scanf("%d",&nodeNum)!=EOF&&nodeNum!=0){
        memset(graph, 0, sizeof(graph));
        scanf("%d %d %d", &from, &to, &edgeNum);
        int u, v, w;
        for (int i = 0; i < edgeNum; ++i){
            scanf("%d %d %d", &u, &v, &w);
            graph[u][v] += w;
            graph[v][u] += w;
        }
        maxFlow = dinic(from, to);
        printf("Network %d\n", NetWorkNum++);
        printf("The bandwidth is %d.\n\n", maxFlow);
    }
    return 0;
}
```

7.9 SegmentTree

```
#define MAXN 1000
int data[MAXN]; //原數據
int st[4 * MAXN]; //線段樹
int tag[4 * MAXN]; //懶標

inline int pull(int l, int r) {
// 隨題目改變sum、max、min
// l、r是左右樹的index
    return st[l] + st[r];
}

void build(int l, int r, int i) {
// 在[l, r]區間建樹，目前根的index為i
    if (l == r) {
        st[i] = data[l];
        return;
    }
    int mid = l + ((r - l) >> 1);
    build(l, mid, i * 2);
```

```
    build(mid + 1, r, i * 2 + 1);
    st[i] = pull(i * 2, i * 2 + 1);
}

int query(int ql, int qr, int l, int r, int i) {
// [ql, qr]是查詢區間,[l, r]是當前節點包含的區間
    if (ql <= l && r <= qr)
        return st[i];
    int mid = l + ((r - l) >> 1);
    if (tag[i]) {
        //如果當前懶標有值則更新左右節點
        st[i * 2] += tag[i] * (mid - l + 1);
        st[i * 2 + 1] += tag[i] * (r - mid);
        tag[i * 2] += tag[i];//下傳懶標至左節點
        tag[i*2+1] += tag[i];//下傳懶標至右節點
        tag[i] = 0;
    }
    int sum = 0;
    if (ql <= mid)
        sum += query(ql, qr, l, mid, i * 2);
    if (qr > mid)
        sum += query(ql, qr, mid + 1, r, i*2+1);
    return sum;
}

void update(int ql,int qr,int l,int r,int i,int c) {
// [ql, qr]是查詢區間,[l, r]是當前節點包含的區間
// c是變化量
    if (ql <= l && r <= qr) {
        st[i] += (r - l + 1) * c;
            //求和,此需乘上區間長度
        tag[i] += c;
        return;
    }
    int mid = l + ((r - l) >> 1);
    if (tag[i] && l != r) {
        //如果當前懶標有值則更新左右節點
        st[i * 2] += tag[i] * (mid - l + 1);
        st[i * 2 + 1] += tag[i] * (r - mid);
        tag[i * 2] += tag[i];//下傳懶標至左節點
        tag[i*2+1] += tag[i];//下傳懶標至右節點
        tag[i] = 0;
    }
    if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
    if (qr > mid) update(ql, qr, mid+1, r, i*2+1, c);
    st[i] = pull(i * 2, i * 2 + 1);
}
//如果是直接改值而不是加值，query與update中的tag與st的
//改值從+=改成=
```

7.10 Nim Game

```
//兩人輪流取銅板，每人每次需在某堆取一枚以上的銅板，
//但不能同時在兩堆取銅板，直到最後，
//將銅板拿光的人贏得此遊戲。

#include <bits/stdc++.h>
#define maxn 23+5
using namespace std;

int SG[maxn];
int visited[1000+5];
int pile[maxn],ans;

void calculateSG(){
    SG[0]=0;
    for(int i=1;i<=maxn;i++){
        int cur=0;
        for(int j=0;j<i;j++)
            for(int k=0;k<=j;k++)
                visited[SG[j]^SG[k]]=i;
        while(visited[cur]==i) cur++;
        SG[i]=cur;
    }
}

int main(){
    calculateSG();
    int Case=0,n;
    while(cin>>n,n){
      ans=0;
      for(int i=1;i<=n;i++) cin>>pile[i];
      for(int i=1;i<=n;i++)
        if(pile[i]&1) ans^=SG[n-i];
      cout<<"Game "<<++Case<<": ";
      if(!ans) cout<<"-1 -1 -1\n";
      else{
        bool flag=0;
        for(int i=1;i<=n;i++){
          if(pile[i]){
            for(int j=i+1;j<=n;j++){
              for(int k=j;k<=n;k++){
                if((SG[n-i]^SG[n-j]^SG[n-k])==ans){
                  cout<<i-1<<" "<<j-1<<" "<<k-1<<endl;
                  flag=1;
                  break;
                }
              }
              if(flag) break;
            }
            if(flag) break;
          }
        }
      }
    }
    return 0;
}

/*
 input
4  1 0 1 100
3  1 0 5
2  2 1
0
 output
Game 1: 0 2 3
Game 2: 0 1 1
Game 3: -1 -1 -1
*/
```

7.11 Trie

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 300000 + 10;
const int mod = 20071027;

int dp[maxn];
int mp[4000*100 + 10][26];
char str[maxn];

struct Trie {
    int seq;
    int val[maxn];

    Trie() {
        seq = 0;
        memset(val, 0, sizeof(val));
        memset(mp, 0, sizeof(mp));
    }

    void insert(char* s, int len) {
        int r = 0;
        for(int i=0; i<len; i++) {
            int c = s[i] - 'a';
            if(!mp[r][c]) mp[r][c] = ++seq;
            r = mp[r][c];
        }
        val[r] = len;
```

```
        return;
    }

    int find(int idx, int len) {
        int result = 0;
        for(int r=0; idx<len; idx++) {
            int c = str[idx] - 'a';
            if(!(r = mp[r][c])) return result;
            if(val[r])
                result = (result + dp[idx + 1]) % mod;
        }
        return result;
    }
};

int main() {
    int n, tc = 1;

    while(~scanf("%s%d", str, &n)) {
        Trie tr;
        int len = strlen(str);
        char word[100+10];

        memset(dp, 0, sizeof(dp));
        dp[len] = 1;

        while(n--) {
            scanf("%s", word);
            tr.insert(word, strlen(word));
        }

        for(int i=len-1; i>=0; i--)
            dp[i] = tr.find(i, len);
        printf("Case %d: %d\n", tc++, dp[0]);
    }
    return 0;
}

/**************
 ****Input****
 * abcd
 * 4
 * a b cd ab
 **************
 ****Output****
 * Case 1: 2
 **************
*/
```

7.12 SPFA

```
struct Edge
{
    int t;
    long long w;
    Edge(){};
    Edge(int _t, long long _w) : t(_t), w(_w) {}
};

bool SPFA(int st) // 平均O(V + E) 最糟O(VE)
{
    vector<int> cnt(n, 0);
    bitset<MXV> inq(0);
    queue<int> q;
    q.push(st);
    dis[st] = 0;
    inq[st] = true;
    while (!q.empty())
    {
        int cur = q.front();
        q.pop();
        inq[cur] = false;
        for (auto &e : G[cur])
        {
            if (dis[e.t] <= dis[cur] + e.w)
                continue;
            dis[e.t] = dis[cur] + e.w;
            if (inq[e.t])
                continue;
            ++cnt[e.t];
            if (cnt[e.t] > n)
                return false; // negtive cycle
            inq[e.t] = true;
            q.push(e.t);
        }
    }
    return true;
}
```

7.13 dijkstra

```
#include<bits/stdc++.h>
#define maxn 50000+5
#define INF 0x3f3f3f3f
using namespace std;

struct edge{
    int v,w;
};

struct Item{
    int u,dis;
    bool operator<(const Item &rhs)const{
        return dis>rhs.dis;
    }
};

vector<edge> G[maxn];
int dist[maxn];

void dijkstra(int s){ // O((V + E)log(E))
    memset(dist,INF,sizeof(dist));
    dist[s]=0;
    priority_queue<Item> pq;
    pq.push({s,0});
    while(!pq.empty()){
        Item now=pq.top();
        pq.pop();
        if(now.dis>dist[now.u]) continue;
        for(edge e:G[now.u]){
            if(dist[e.v]>dist[now.u]+e.w){
                dist[e.v]=dist[now.u]+e.w;
                pq.push({e.v,dist[e.v]});
            }
        }
    }
}

int main(){
    int t,cas=1;
    cin>>t;
    while(t--){
        int n,m,s,t;
        cin>>n>>m>>s>>t;
        for(int i=0;i<=n;i++) G[i].clear();
        int u,v,w;
        for(int i=0;i<m;i++){
            cin>>u>>v>>w;
            G[u].push_back({v,w});
            G[v].push_back({u,w});
        }
        dijkstra(s);
        cout<<"Case #"<<cas++<<": ";
        if(dist[t]==INF) cout<<"unreachable\n";
        else cout<<dist[t]<<endl;
    }
}
```

7.14 SCC Tarjan

```cpp
//Strongly Connected Components
//Tarjan O(V + E)
int dfn[N], low[N], dfncnt, sk[N], in_stack[N], tp;
//dfn[u]: dfs時u被visited的順序
//low[u]: 在u的dfs子樹中能回到最早已在stack中的節點
int scc[N], sc;//節點 u 所在 SCC 的編號
int sz[N];//強連通 u 的大小

void tarjan(int u) {
    low[u] = dfn[u] = ++dfncnt, s[++tp] = u,
        in_stack[u] = 1;
    for (int i = h[u]; i; i = e[i].nex) {
        const int &v = e[i].t;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_stack[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ++sc;
        while (s[tp] != u) {
            scc[s[tp]] = sc;
            sz[sc]++;
            in_stack[s[tp]] = 0;
            --tp;
        }
        scc[s[tp]] = sc;
        sz[sc]++;
        in_stack[s[tp]] = 0;
        --tp;
    }
}
```

7.15 SCC Kosaraju

```cpp
//做兩次dfs, O(V + E)
//g 是原圖，g2 是反圖
//s是dfs離開的節點
void dfs1(int u) {
    vis[u] = true;
    for (int v : g[u])
        if (!vis[v]) dfs1(v);
    s.push_back(u);
}

void dfs2(int u) {
    group[u] = sccCnt;
    for (int v : g2[u])
        if (!group[v]) dfs2(v);
}

void kosaraju() {
    sccCnt = 0;
    for (int i = 1; i <= n; ++i)
        if (!vis[i]) dfs1(i);
    for (int i = n; i >= 1; --i)
        if (!group[s[i]]) {
            ++sccCnt;
            dfs2(s[i]);
        }
}
```

7.16 ArticulationPoints Tarjan

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> G;
int N;
int timer;
bool visited[105];
int visTime[105]; // 第一次visit的時間
int low[105];
// 最小能回到的父節點(不能是自己的parent)的visTime
int res;
//求割點數量
void tarjan(int u, int parent) {
    int child = 0;
    bool isCut = false;
    visited[u] = true;
    visTime[u] = low[u] = ++timer;
    for (int v: G[u]) {
        if (!visited[v]) {
            ++child;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (parent != -1 && low[v] >= visTime[u])
                isCut = true;
        }
        else if (v != parent)
            low[u] = min(low[u], visTime[v]);
    }
    //If u is root of DFS tree->有兩個以上的children
    if (parent == -1 && child >= 2)
        isCut = true;
    if (isCut)
        ++res;
}

int main()
{
    char input[105];
    char* token;
    while (scanf("%d", &N) != EOF && N)
    {
        G.assign(105, vector<int>());
        memset(visited, false, sizeof(visited));
        memset(low, 0, sizeof(low));
        memset(visTime, 0, sizeof(visited));
        timer = 0;
        res = 0;
        getchar(); // for \n
        while (fgets(input, 105, stdin))
        {
            if (input[0] == '0')
                break;
            int size = strlen(input);
            input[size - 1] = '\0';
            --size;
            token = strtok(input, " ");
            int u = atoi(token);
            int v;
            while (token = strtok(NULL, " "))
            {
                v = atoi(token);
                G[u].emplace_back(v);
                G[v].emplace_back(u);
            }
        }
        tarjan(1, -1);
        printf("%d\n", res);
    }
    return 0;
}
```

7.17 最小樹狀圖

```
  定義
有向圖上的最小生成樹 (Directed Minimum Spanning Tree)
稱為最小樹形圖。
常用的演算法是朱劉演算法 (也稱為Edmonds 演算法) ，
可以在O(nm)時間內解決最小樹形圖問題。

```

```
  流程
1. 對於每個點，選擇它入度最小的那條邊
2. 如果沒有環，演算法終止；
   否則進行縮環並更新其他點到環的距離。

bool solve() {
  ans = 0;
  int u, v, root = 0;
  for (;;) {
    f(i, 0, n) in[i] = 1e100;
    f(i, 0, m) {
      u = e[i].s;
      v = e[i].t;
      if (u != v && e[i].w < in[v]) {
        in[v] = e[i].w;
        pre[v] = u;
      }
    }
    f(i, 0, m) if(i!=root && in[i]>1e50) return 0;
    int tn = 0;
    memset(id, -1, sizeof id);
    memset(vis, -1, sizeof vis);
    in[root] = 0;
    f(i, 0, n) {
      ans += in[i];
      v = i;
      while(vis[v]!=i&&id[v]==-1&&v!=root){
        vis[v] = i;
        v = pre[v];
      }
      if (v != root && id[v] == -1) {
        for(int u=pre[v];u!=v;u=pre[u]) id[u]=tn;
        id[v] = tn++;
      }
    }
    if (tn == 0) break;
    f(i, 0, n) if (id[i] == -1) id[i] = tn++;
    f(i, 0, m) {
      u = e[i].s;
      v = e[i].t;
      e[i].s = id[u];
      e[i].t = id[v];
      if (e[i].s != e[i].t) e[i].w -= in[v];
    }
    n = tn;
    root = id[root];
  }
  return ans;
}



  Tarjan 的DMST 演算法
Tarjan 提出了一種能夠在
O(m+nlog n)時間內解決最小樹形圖問題的演算法。

  流程
Tarjan 的演算法分為收縮與伸展兩個過程。
接下來先介紹收縮的過程。
我們要假設輸入的圖是滿足強連通的，
如果不滿足那就加入 O(n) 條邊使其滿足，
並且這些邊的邊權是無窮大的。

我們需要一個堆存儲結點的入邊編號，入邊權值，
結點總代價等相關信息，由於後續過程中會有堆的合併操作，
這裡採用左偏樹 與並查集實現。
演算法的每一步都選擇一個任意結點v，
需要保證v不是根節點，並且在堆中沒有它的入邊。
再將v的最小入邊加入到堆中，
如果新加入的這條邊使堆中的邊形成了環，
那麼將構成環的那些結點收縮，
我們不妨將這些已經收縮的結點命名為超級結點，
再繼續這個過程，如果所有的頂點都縮成了超級結點，
那麼收縮過程就結束了。
整個收縮過程結束後會得到一棵收縮樹，
之後就會對它進行伸展操作。

堆中的邊總是會形成一條路徑v0 <- v1<- ... <- vk，
由於圖是強連通的，這個路徑必然存在，
並且其中的 vi 可能是最初的單一結點，
也可能是壓縮後的超級結點。

最初有 v0=a，其中 a 是圖中任意的一個結點，
每次都選擇一條最小入邊 vk <- u，
如果 u 不是v0,v1,...,vk中的一個結點，
那麼就將結點擴展到 v k+1=u。
如果 u 是他們其中的一個結點 vi，
那麼就找到了一個關於 vi <- ... <- vk <- vi的環，
再將他們收縮為一個超級結點c。

向隊列 P 中放入所有的結點或超級結點，
並初始選擇任一節點 a，只要佇列不為空，就進行以下步驟：

選擇 a 的最小入邊，保證不存在自環，
並找到另一頭的結點 b。
如果結點b沒有被記錄過說明未形成環，
令 a <- b，繼續目前操作尋找環。

如果 b 被記錄過了，就表示出現了環。
總結點數加一，並將環上的所有結點重新編號，對堆進行合併，
以及結點/超級結點的總權值的更新。
更新權值操作就是將環上所有結點的入邊都收集起來，
並減去環上入邊的邊權。


#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 102
#define INF 0x3f3f3f3f

struct UnionFind {
  int fa[maxn << 1];
  UnionFind() { memset(fa, 0, sizeof(fa)); }
  void clear(int n) {
    memset(fa + 1, 0, sizeof(int) * n);
  }
  int find(int x) {
    return fa[x] ? fa[x] = find(fa[x]) : x;
  }
  int operator[](int x) { return find(x); }
};

struct Edge {
  int u, v, w, w0;
};

struct Heap {
  Edge *e;
  int rk, constant;
  Heap *lch, *rch;

  Heap(Edge *_e):
    e(_e),rk(1),constant(0),lch(NULL),rch(NULL){}

  void push() {
    if (lch) lch->constant += constant;
    if (rch) rch->constant += constant;
    e->w += constant;
    constant = 0;
  }
};

Heap *merge(Heap *x, Heap *y) {
  if (!x) return y;
  if (!y) return x;
  if(x->e->w + x->constant > y->e->w + y->constant)
    swap(x, y);
  x->push();
  x->rch = merge(x->rch, y);
```

```
  if (!x->lch || x->lch->rk < x->rch->rk)
    swap(x->lch, x->rch);
  if (x->rch)
    x->rk = x->rch->rk + 1;
  else
    x->rk = 1;
  return x;
}

Edge *extract(Heap *&x) {
  Edge *r = x->e;
  x->push();
  x = merge(x->lch, x->rch);
  return r;
}

vector<Edge> in[maxn];
int n, m, fa[maxn << 1], nxt[maxn << 1];
Edge *ed[maxn << 1];
Heap *Q[maxn << 1];
UnionFind id;

void contract() {
  bool mark[maxn << 1];
  //將圖上的每一個節點與其相連的那些節點進行記錄
  for (int i = 1; i <= n; i++) {
    queue<Heap *> q;
    for (int j = 0; j < in[i].size(); j++)
      q.push(new Heap(&in[i][j]));
    while (q.size() > 1) {
      Heap *u = q.front();
      q.pop();
      Heap *v = q.front();
      q.pop();
      q.push(merge(u, v));
    }
    Q[i] = q.front();
  }
  mark[1] = true;
  for(int a=1,b=1,p;Q[a];b=a,mark[b]=true){
    //尋找最小入邊以及其端點，保證無環
    do {
      ed[a] = extract(Q[a]);
      a = id[ed[a]->u];
    } while (a == b && Q[a]);
    if (a == b) break;
    if (!mark[a]) continue;
    //對發現的環進行收縮，以及環內的節點重新編號，
    //總權值更新
    for (a = b, n++; a != n; a = p) {
      id.fa[a] = fa[a] = n;
      if (Q[a]) Q[a]->constant -= ed[a]->w;
      Q[n] = merge(Q[n], Q[a]);
      p = id[ed[a]->u];
      nxt[p == n ? b : p] = a;
    }
  }
}

ll expand(int x, int r);
ll expand_iter(int x) {
  ll r = 0;
  for(int u=nxt[x];u!=x;u=nxt[u]){
    if (ed[u]->w0 >= INF)
      return INF;
    else
      r+=expand(ed[u]->v,u)+ed[u]->w0;
  }
  return r;
}

ll expand(int x, int t) {
  ll r = 0;
  for (; x != t; x = fa[x]) {
    r += expand_iter(x);
    if (r >= INF) return INF;
  }
  return r;
}

void link(int u, int v, int w) {
  in[v].push_back({u, v, w, w});
}

int main() {
  int rt;
  scanf("%d %d %d", &n, &m, &rt);
  for (int i = 0; i < m; i++) {
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    link(u, v, w);
  }
  //保證強連通
  for (int i = 1; i <= n; i++)
    link(i > 1 ? i - 1 : n, i, INF);
  contract();
  ll ans = expand(rt, n);
  if (ans >= INF)
    puts("-1");
  else
    printf("%lld\n", ans);
  return 0;
}
```

7.18 凸包

```
/* ***************************************************
 * 問題：在平面上給定多區域，再給定多點(x, y)，
 * 判斷有落點的區域(destroyed)的面積總和。
 * ***************************************************/
#include <bits/stdc++.h>
using namespace std;

const int maxn = 500 + 10;
const int maxCoordinate = 500 + 10;

struct Point {
    int x, y;
};

int n;
bool destroyed[maxn] = {};
Point arr[maxn];
vector<Point> polygons[maxn];

void scanAndSortPoints() {
    int minX = maxCoordinate, minY = maxCoordinate;
    for(int i=0; i<n; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        arr[i] = (Point){x, y};
        if(y < minY || (y == minY && x < minX)) {
    //  If there are floating points, use:
    //  if(y<minY || (fabs(y-minY)<eps && x<minX)) {
            minX = x, minY = y;
        }
    }
    sort(arr, arr+n, [minX, minY](Point& a, Point& b){
        double theta1 = atan2(a.y - minY, a.x - minX);
        double theta2 = atan2(b.y - minY, b.x - minX);
        return theta1 < theta2;
    });
    return;
}

//  returns u(AB) x v(AC)
int cross(Point& A, Point& B, Point& C) {
    int u[2] = {B.x - A.x, B.y - A.y};
    int v[2] = {C.x - A.x, C.y - A.y};
    return (u[0] * v[1]) - (u[1] * v[0]);
}

```

```cpp
// size of arr >= 3
// st = the stack using vector, m = index of the top
vector<Point> convex_hull() {
    vector<Point> st(arr, arr+3);
    for(int i=3, m=2; i<n; i++, m++) {
        while(m >= 2) {
            if(cross(st[m], st[m-1], arr[i]) < 0)
                break;
            st.pop_back();
            m--;
        }
        st.push_back(arr[i]);
    }
    return st;
}

bool inPolygon(vector<Point>& vec, Point p) {
    vec.push_back(vec[0]);
    for(int i=1; i<vec.size(); i++) {
        if(cross(vec[i-1], vec[i], p) < 0) {
            vec.pop_back();
            return false;
        }
    }
    vec.pop_back();
    return true;
}

/* ***************************************************
 *        | x1    x2    x3    x4    x5          xn |
 *  A =   |     x     x     x     x     x ... x     | x 0.5
 *        | y1    y2    y3    y4    y5          yn |
 ***************************************************/
double calculateArea(vector<Point>& v) {
    v.push_back(v[0]);
    double result = 0.0;
    for(int i=1; i<v.size(); i++)
        result += v[i-1].x*v[i].y - v[i-1].y*v[i].x;
    v.pop_back();
    return result / 2.0;
}

int main() {
    int p = 0;
    while(~scanf("%d", &n) && (n != -1)) {
        scanAndSortPoints();
        polygons[p++] = convex_hull();
    }

    int x, y;
    double result = 0.0;
    while(~scanf("%d%d", &x, &y)) {
        for(int i=0; i<p; i++) {
            if(inPolygon(polygons[i], (Point){x, y}))
                destroyed[i] = true;
        }
    }
    for(int i=0; i<p; i++) {
        if(destroyed[i])
            result += calculateArea(polygons[i]);
    }
    printf("%.2lf\n", result);
    return 0;
}
```

8 動態規劃

8.1 LCS 和 LIS

```
//最長共同子序列(LCS)
給定兩序列 A,B ，求最長的序列 C ，
 C 同時為 A,B 的子序列。

//最長遞增子序列 (LIS)
 給你一個序列 A ，求最長的序列 B ，
  B 是一個 (非) 嚴格遞增序列，且為 A 的子序列。

//LCS 和 LIS 題目轉換
LIS 轉成 LCS
    1. A 為原序列， B=sort(A)
    2. 對 A,B 做 LCS
LCS 轉成 LIS
    1. A, B 為原本的兩序列
    2. 最 A 序列作編號轉換，將轉換規則套用在 B
    3. 對 B 做 LIS
    4. 重複的數字在編號轉換時後要變成不同的數字，
       越早出現的數字要越小
    5. 如果有數字在 B 裡面而不在 A 裡面，
       直接忽略這個數字不做轉換即可
```

9 Section2

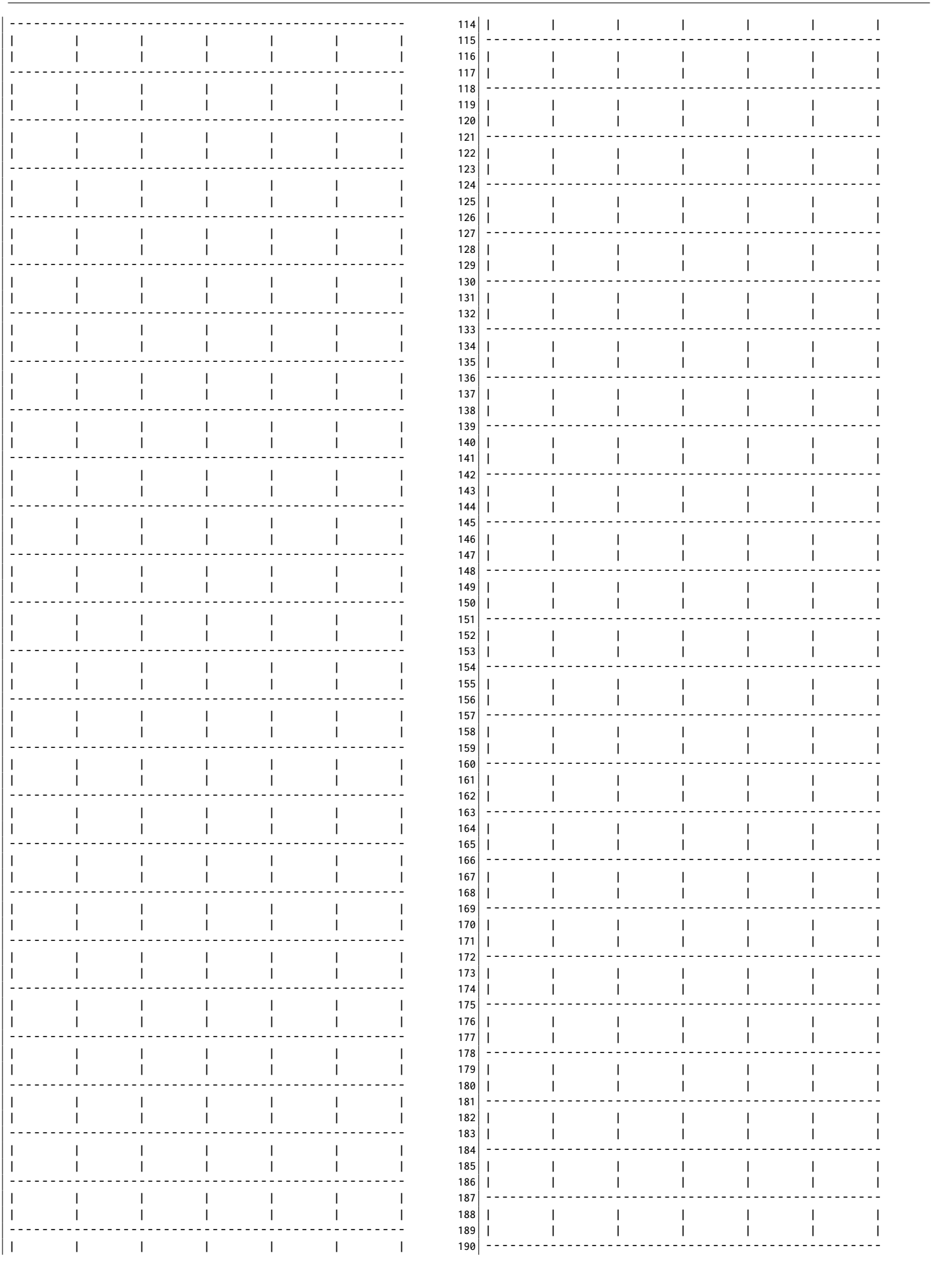
9.1 thm

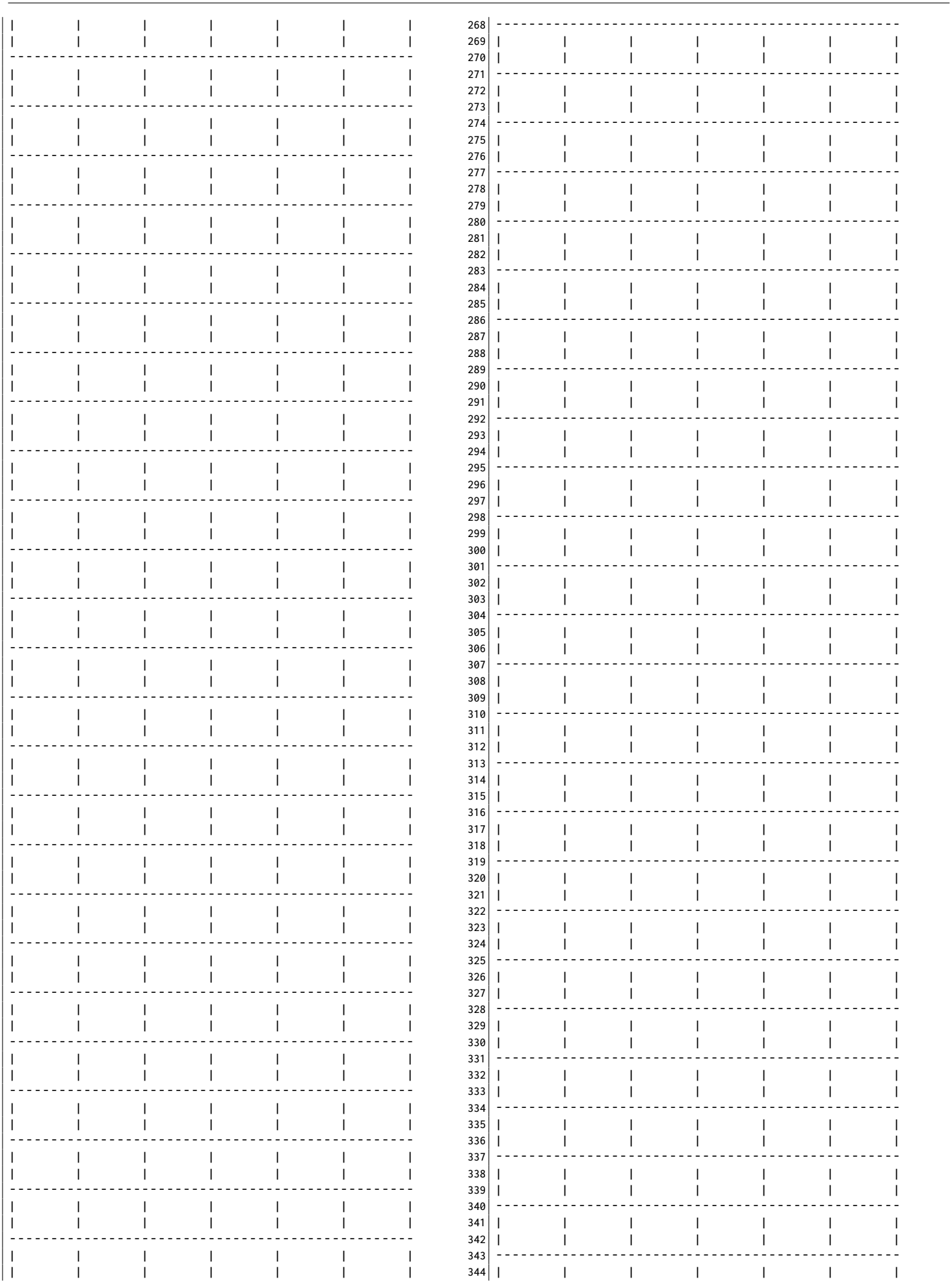
- 中文測試
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\binom{x}{y} = \frac{x!}{y!(x-y)!}$
- $\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$
- $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

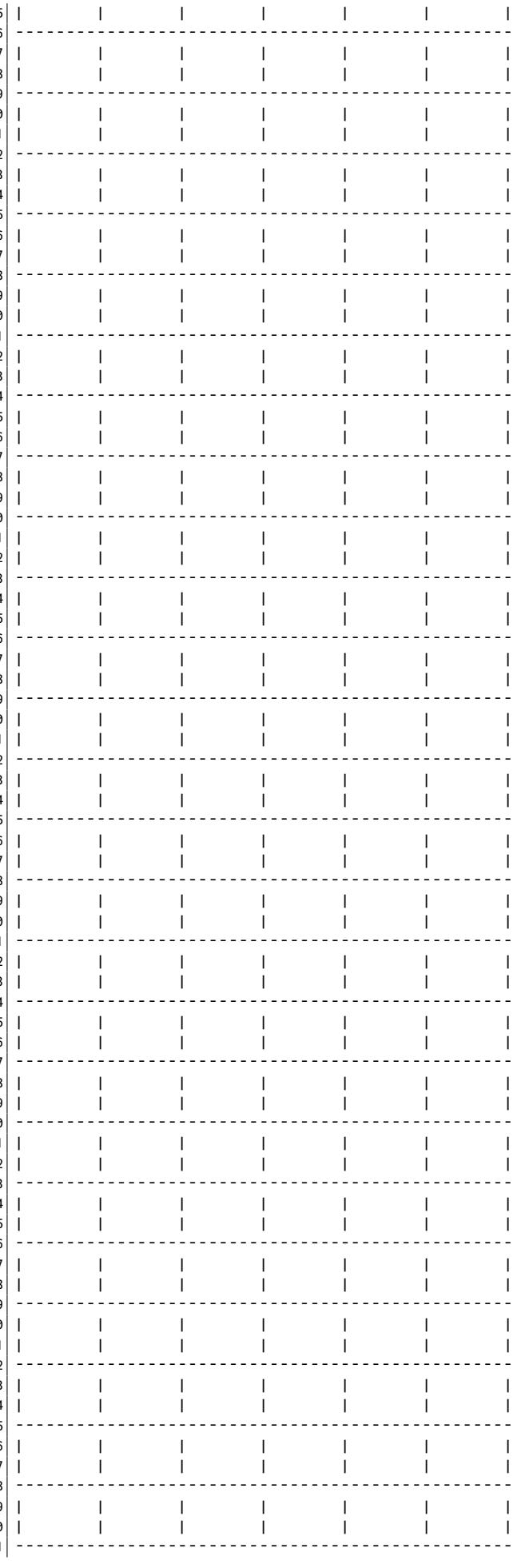
10 dp 表格

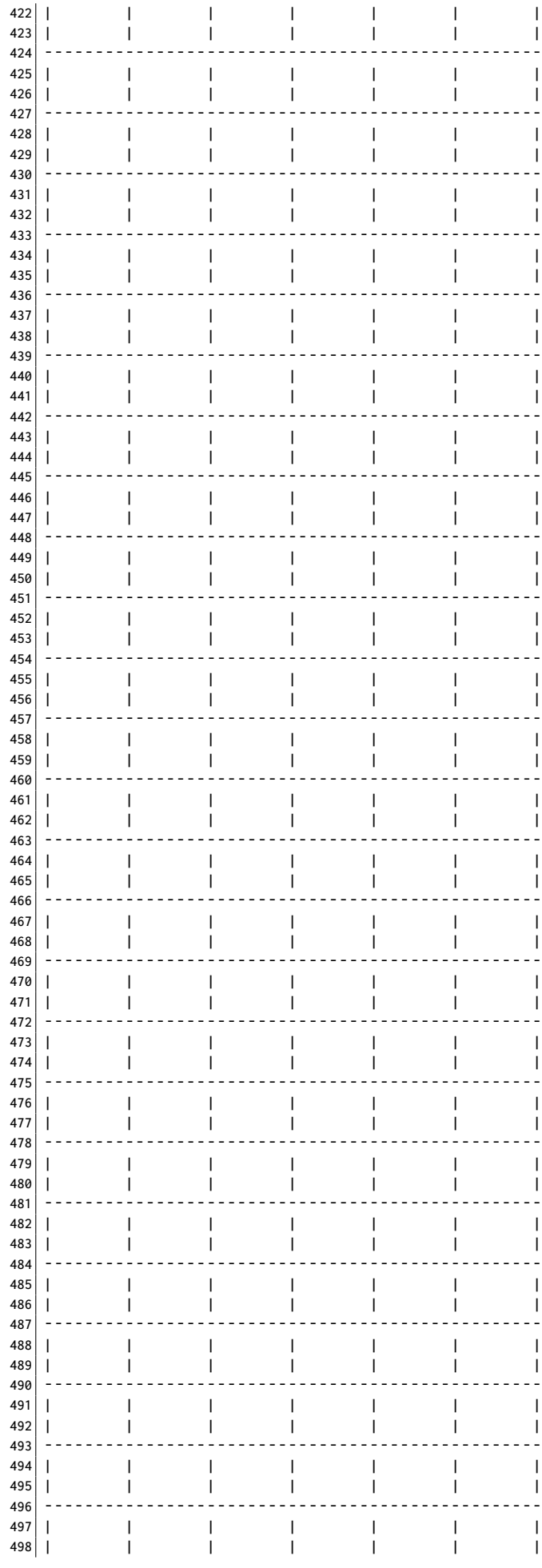
10.1 DPlist

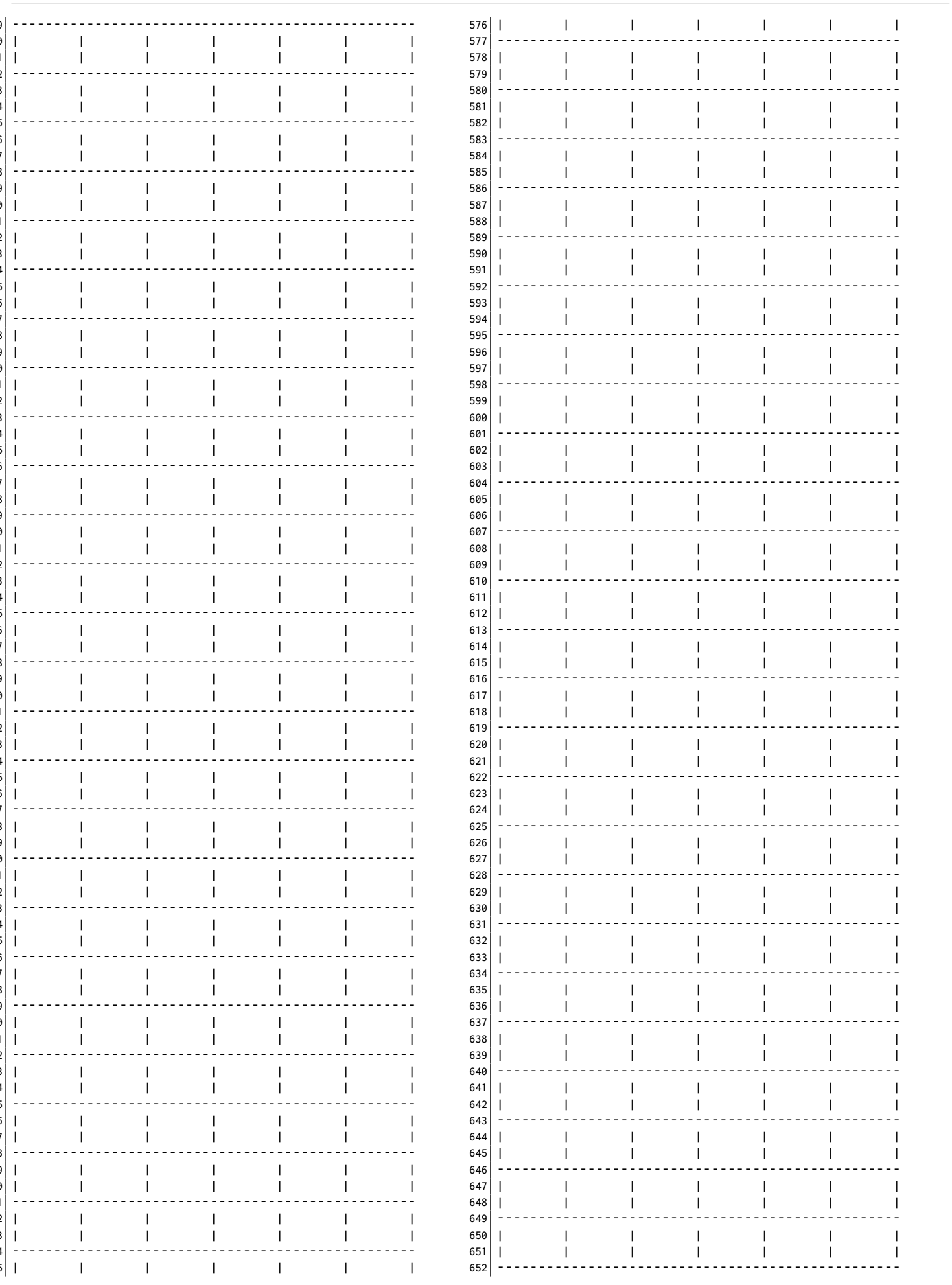
```
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
-------------------------------------------------
|       |       |       |       |       |       |
|       |       |       |       |       |       |
```

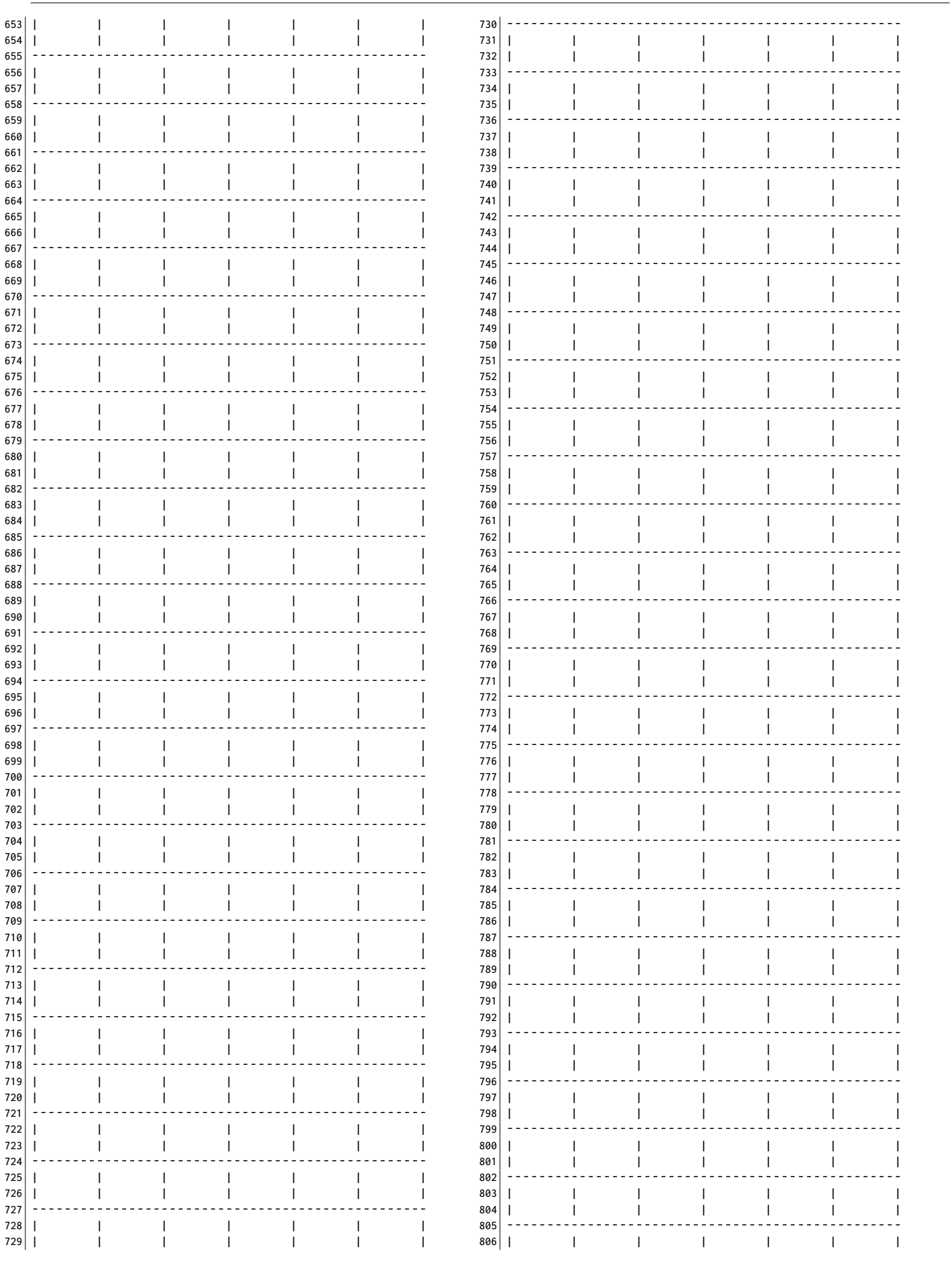


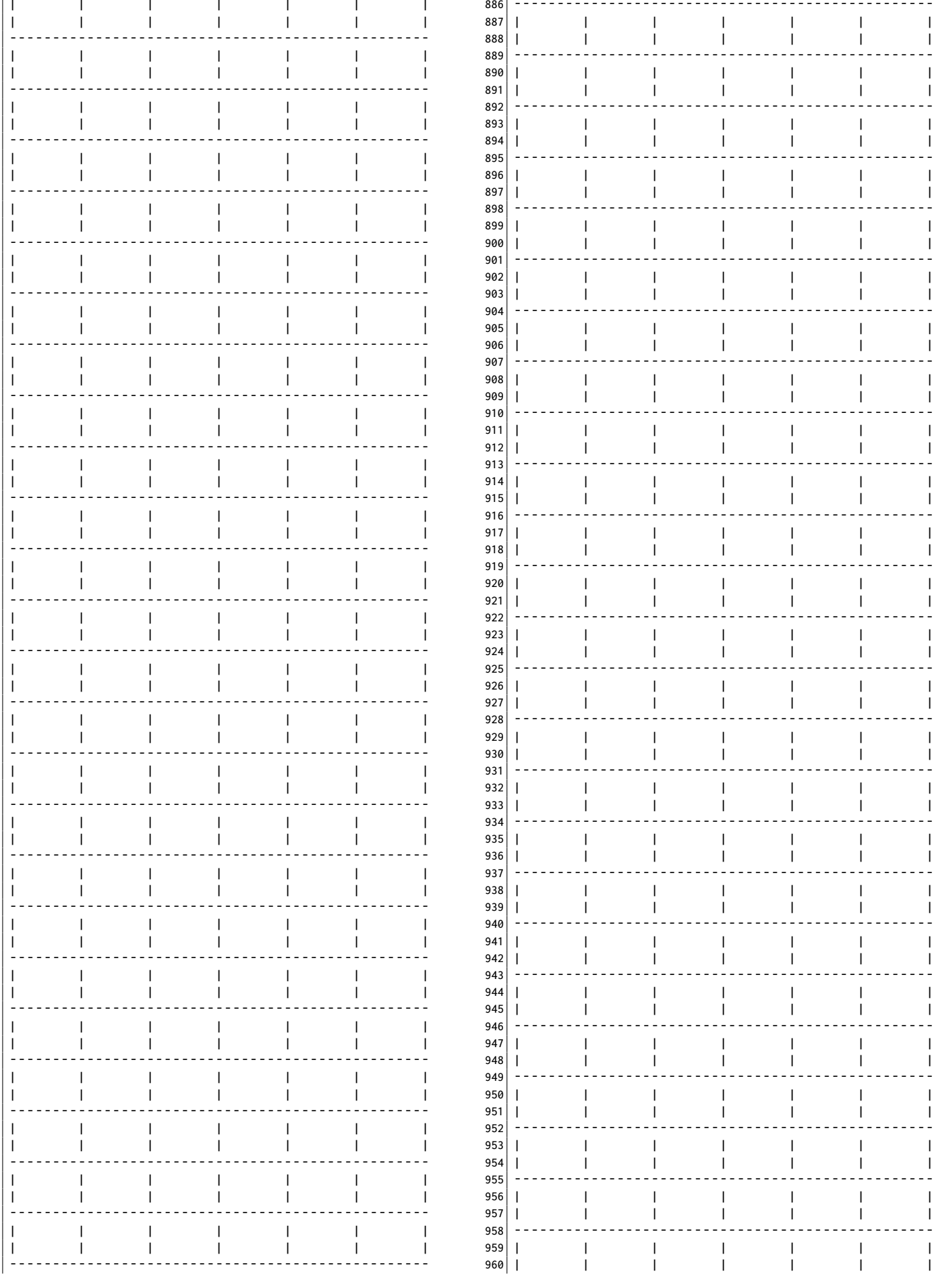


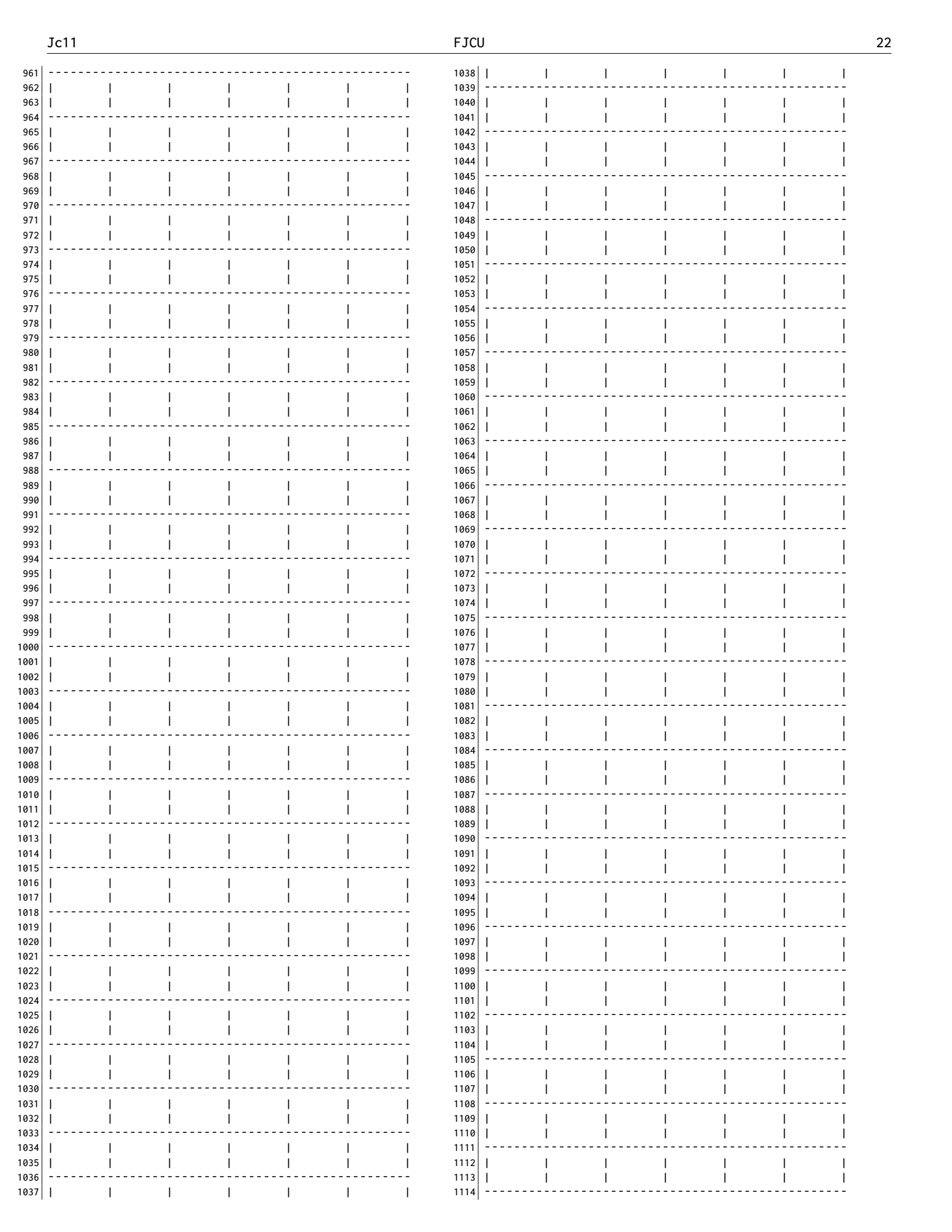


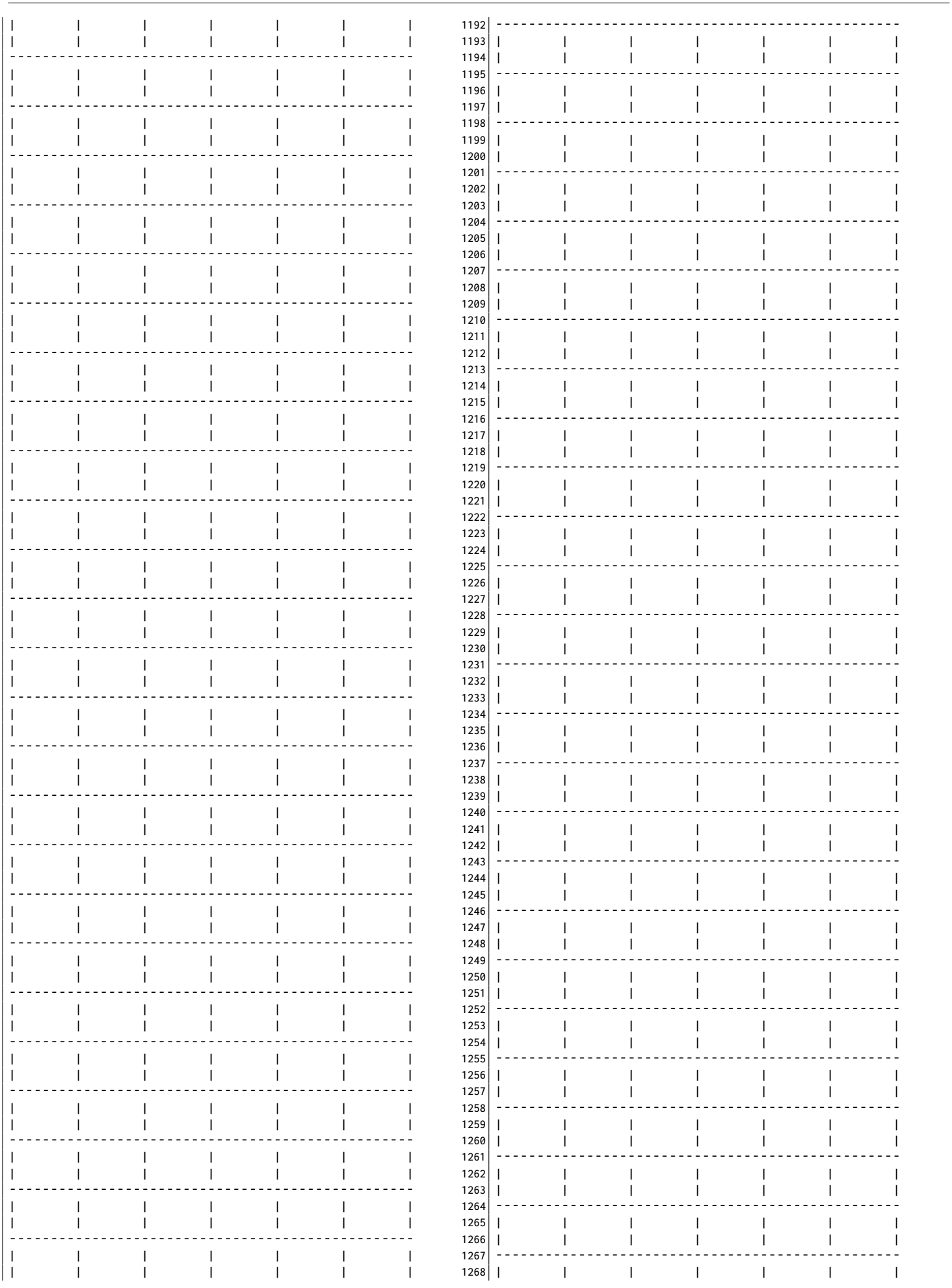


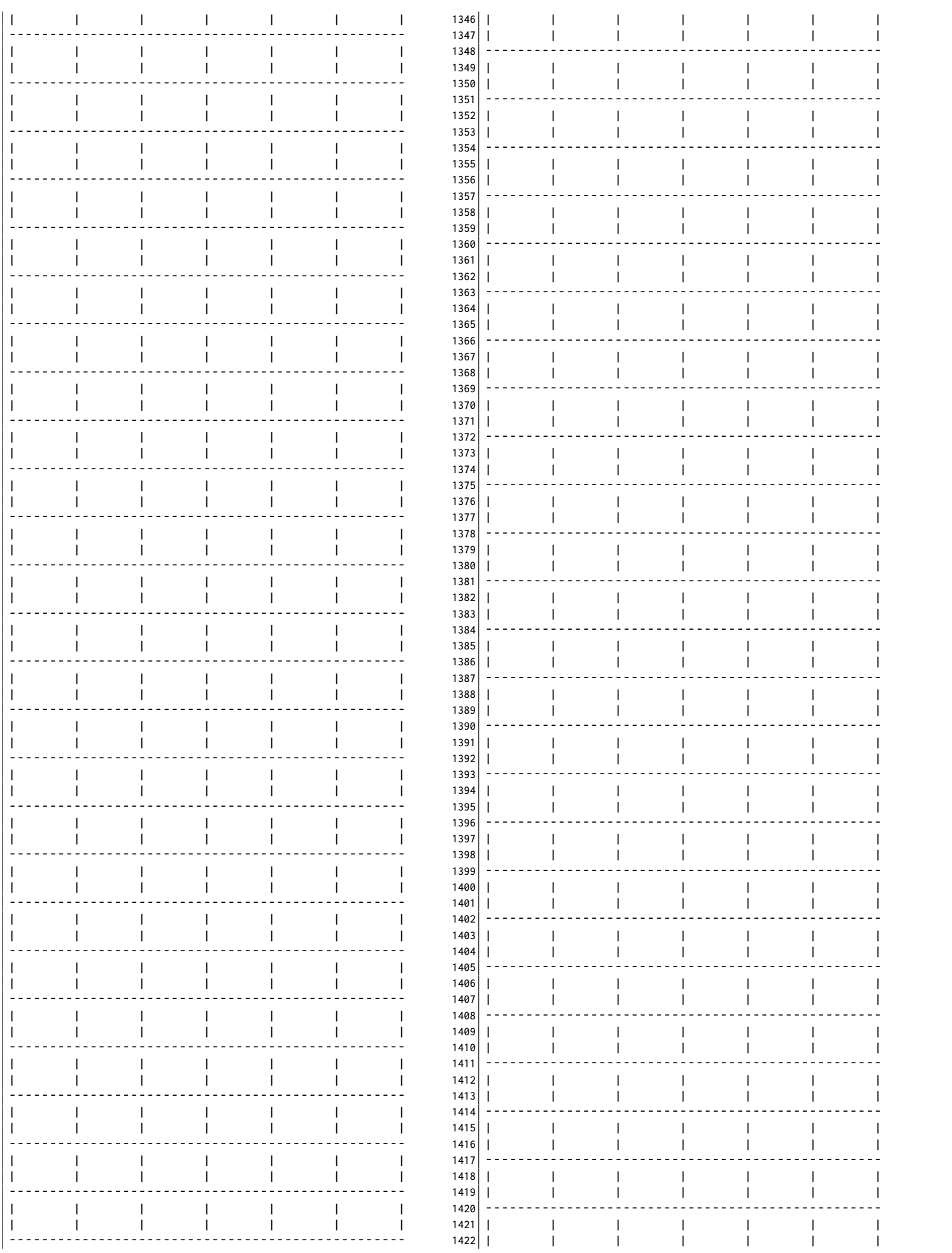












11 slogan

11.1 slogan