

Contents

1	ubuntu	1
1.1	run	1
1.2	cp.sh	1
2	Basic	1
2.1	ascii	1
2.2	limits	1
3	字串	1
3.1	最長迴文子字串	1
3.2	stringstream	2
4	STL	2
4.1	priority_queue	2
4.2	deque	2
4.3	map	2
4.4	unordered_map	3
4.5	set	3
4.6	multiset	3
4.7	unordered_set	3
4.8	單調隊列	3
5	sort	3
5.1	大數排序	3
6	math	4
6.1	質數與因數	4
6.2	快速冪	4
6.3	歐拉函數	5
6.4	atan	5
7	algorithm	5
7.1	basic	5
7.2	binary search	5
7.3	prefix sum	5
7.4	差分	6
7.5	greedy	6
7.6	floyd warshall	7
7.7	dinic	8
7.8	SegmentTree	8
7.9	Nim Game	9
7.10	Trie	9
7.11	SPFA	10
7.12	dijkstra	10
8	動態規劃	10
8.1	LCS 和 LIS	10
9	Section2	11
9.1	thm	11

1 ubuntu

1.1 run

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

1.2 cp.sh

```
1 #!/bin/bash
2 clear
3 g++ $1.cpp -DDBG -o $1
4 if [[ "$?" == "0" ]]; then
5     echo Running
6     ./$1 < $1.in > $1.out
7     echo END
8 fi
```

2 Basic

2.1 ascii

	int	char	int	char	int	char
1	32		64	@	96	`
2	33	!	65	A	97	a
3	34	"	66	B	98	b
4	35	#	67	C	99	c
5	36	\$	68	D	100	d
6	37	%	69	E	101	e
7	38	&	70	F	102	f
8	39	'	71	G	103	g
9	40	(72	H	104	h
10	41)	73	I	105	i
11	42	*	74	J	106	j
12	43	+	75	K	107	k
13	44	,	76	L	108	l
14	45	-	77	M	109	m
15	46	.	78	N	110	n
16	47	/	79	O	111	o
17	48	0	80	P	112	p
18	49	1	81	Q	113	q
19	50	2	82	R	114	r
20	51	3	83	S	115	s
21	52	4	84	T	116	t
22	53	5	85	U	117	u
23	54	6	86	V	118	v
24	55	7	87	W	119	w
25	56	8	88	X	120	x
26	57	9	89	Y	121	y
27	58	:	90	Z	122	z
28	59	;	91	[123	{
29	60	<	92	\	124	
30	61	=	93]	125	}
31	62	>	94	^	126	~
32	63	?	95	_		

2.2 limits

	[Type]	[size]	[range]
1	char	1	127 to -128
2	signed char	1	127 to -128
3	unsigned char	1	0 to 255
4	short	2	32767 to -32768
5	int	4	2147483647 to -2147483648
6	unsigned int	4	0 to 4294967295
7	long	4	2147483647 to -2147483648
8	unsigned long	4	0 to 18446744073709551615
9	long long	8	
10		9223372036854775807 to -9223372036854775808	
11	double	8	1.79769e+308 to 2.22507e-308
12	long double	16	1.18973e+4932 to 3.3621e-4932
13	float	4	3.40282e+38 to 1.17549e-38
14	unsigned long long	8	0 to 18446744073709551615
15	string	32	

3 字串

3.1 最長迴文子字串

```
1 #include<bits/stdc++.h>
2 #define T(x) ((x)%2 ? s[(x)/2] : '. ')
3 using namespace std;
4
5 string s;
6 int n;
7
8 int ex(int l,int r){
9     int i=0;
```

```

10 while(l-i>=0&&r+i<n&&T(l-i)==T(r+i)) i++;
11 return i;
12 }
13
14 int main(){
15     cin>>s;
16     n=2*s.size()+1;
17     int mx=0;
18     int center=0;
19     vector<int> r(n);
20     int ans=1;
21     r[0]=1;
22     for(int i=1;i<n;i++){
23         int ii=center-(i-center);
24         int len=mx-i+1;
25         if(i>mx){
26             r[i]=ex(i,i);
27             center=i;
28             mx=i+r[i]-1;
29         }
30         else if(r[ii]==len){
31             r[i]=len+ex(i-len,i+len);
32             center=i;
33             mx=i+r[i]-1;
34         }
35         else r[i]=min(r[ii],len);
36         ans=max(ans,r[i]);
37     }
38     cout<<ans-1<<"\n";
39     return 0;
40 }

```

3.2 stringstream

```

1 string s,word;
2 stringstream ss;
3 getline(cin,s);
4 ss<<s;
5 while(ss>>word) cout<<word<<endl;

```

4 STL

4.1 priority_queue

```

1 priority_queue: 優先隊列，資料預設由大到小排序。
2
3 讀取優先權最高的值：
4     x = pq.top();
5     pq.pop(); //讀取後刪除
6 判斷是否為空的priority_queue：
7     pq.empty() //回傳true
8     pq.size() //回傳0
9 如需改變priority_queue的優先權定義：
10    priority_queue<T> pq; //預設由大到小
11    priority_queue<T, vector<T>, greater<T>> > pq;
12    //改成由小到大
13    priority_queue<T, vector<T>, cmp> pq; //cmp

```

4.2 deque

```

1 deque 是 C++ 標準模板函式庫
2   (Standard Template Library, STL)
3   中的雙向佇列容器 (Double-ended Queue) ,
4   跟 vector 相似，不過在 vector
5   中若是要添加新元素至開端，
6   其時間複雜度為 O(N)，但在 deque 中則是 O(1)。
7   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。

```

```

8 dq.push_back() //在 deque 的最尾端新增元素
9 dq.push_front() //在 deque 的開頭新增元素
10 dq.pop_back() //移除 deque 最尾端的元素
11 dq.pop_front() //移除 deque 最開頭的元素
12 dq.back() //取出 deque 最尾端的元素
13 dq.front() //回傳 deque 最開頭的元素
14 dq.insert()
15 dq.insert(position, n, val)
16     position: 插入元素的 index 值
17     n: 元素插入次數
18     val: 插入的元素值
19 dq.erase()
20 //刪除元素，需要使用迭代器指定刪除的元素或位置，
21 //同時也會返回指向刪除元素下一元素的迭代器。
22 dq.clear() //清空整個 deque 佇列。
23 dq.size() //檢查 deque 的尺寸
24 dq.empty() //如果 deque 佇列為空返回 1；
25 //若是存在任何元素，則返回0
26 dq.begin() //返回一個指向 deque 開頭的迭代器
27 dq.end() //指向 deque 結尾，
28 //不是最後一個元素，
29 //而是最後一個元素的下一個位置

```

4.3 map

```

1 map: 存放 key-value pairs 的映射資料結構，
2     會按 key 由小到大排序。
3 元素存取
4 operator[]: 存取指定的[i]元素的資料
5
6 迭代器
7 begin(): 回傳指向map頭部元素的迭代器
8 end(): 回傳指向map末尾的迭代器
9 rbegin(): 回傳一個指向map尾部的反向迭代器
10 rend(): 回傳一個指向map頭部的反向迭代器
11
12 遍歷整個map時，利用iterator操作：
13 取key: it->first 或 (*it).first
14 取value: it->second 或 (*it).second
15
16 容量
17 empty(): 檢查容器是否為空，空則回傳true
18 size(): 回傳元素數量
19 max_size(): 回傳可以容納的最大元素個數
20
21 修改器
22 clear(): 刪除所有元素
23 insert(): 插入元素
24 erase(): 刪除一個元素
25 swap(): 交換兩個map
26
27 查找
28 count(): 回傳指定元素出現的次數
29 find(): 查找一個元素
30
31 //實作範例
32 #include <bits/stdc++.h>
33 using namespace std;
34 int main(){
35     //declaration container and iterator
36     map<string, string> mp;
37     map<string, string>::iterator iter;
38     map<string, string>::reverse_iterator iter_r;
39
40     //insert element
41     mp.insert(pair<string, string>
42         ("r000", "student_zero"));
43     mp["r123"] = "student_first";
44     mp["r456"] = "student_second";
45
46     //traversal

```

```

47 for(iter=mp.begin();iter!=mp.end();iter++)
48     cout<<iter->first<<" "
49         <<iter->second<<endl;
50 for(iter_r=mp.rbegin();iter_r!=mp.rend();iter_r++)
51     cout<<iter_r->first<<"
52         "<<iter_r->second<<endl;
53
54 //find and erase the element
55 iter=mp.find("r123");
56 mp.erase(iter);
57 iter=mp.find("r123");
58 if(iter!=mp.end())
59     cout<<"Find, the value is "
60         <<iter->second<<endl;
61 else cout<<"Do not Find"<<endl;
62 return 0;
63 }

```

4.4 unordered_map

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

4.5 set

1 set：集合，去除重複的元素，資料由小到大排序。
 2
 3 取值：使用iterator
 4 x = *st.begin();
 5 // set中的第一個元素(最小的元素)。
 6 x = *st.rbegin();
 7 // set中的最後一個元素(最大的元素)。
 8
 9 判斷是否為空的set：
 10 st.empty() 回傳true
 11 st.size() 回傳零
 12
 13 常用來搭配的member function：
 14 st.count(x);
 15 auto it = st.find(x);
 16 // binary search, $O(\log(N))$
 17 auto it = st.lower_bound(x);
 18 // binary search, $O(\log(N))$
 19 auto it = st.upper_bound(x);
 20 // binary search, $O(\log(N))$

4.6 multiset

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

4.7 unordered_set

1 unordered_set 的實作方式通常是用雜湊表(hash table)，
 2 資料插入和查詢的時間複雜度很低，為常數級別 $O(1)$ ，
 3 相對的代價是消耗較多的記憶體，空間複雜度較高，
 4 無自動排序功能。
 5
 6 unordered_set 判斷元素是否存在

```

7 unordered_set<int> myunordered_set;
8 myunordered_set.insert(2);
9 myunordered_set.insert(4);
10 myunordered_set.insert(6);
11 cout << myunordered_set.count(4) << "\n"; // 1
12 cout << myunordered_set.count(8) << "\n"; // 0

```

4.8 單調隊列

```

1 //單調隊列
2 "如果一個選手比你小還比你強，你就可以退役了。"--單調隊列
3
4 example
5
6 給出一個長度為 n 的數組，
7 輸出每 k 個連續的數中的最大值和最小值。
8
9 #include <bits/stdc++.h>
10 #define maxn 1000100
11 using namespace std;
12 int q[maxn], a[maxn];
13 int n, k;
14
15 void getmin() {
16     // 得到這個隊列裡的最小值，直接找到最後的就行了
17     int head=0,tail=0;
18     for(int i=1;i<=n;i++) {
19         while(head<=tail&&a[q[tail]]>=a[i]) tail--;
20         q[++tail]=i;
21     }
22     for(int i=k;i<=n;i++) {
23         while(head<=tail&&a[q[tail]]>=a[i]) tail--;
24         q[++tail]=i;
25         while(q[head]<=i-k) head++;
26         cout<<a[q[head]]<<" ";
27     }
28     cout<<endl;
29 }
30
31 void getmax() { // 和上面同理
32     int head=0,tail=0;
33     for(int i=1;i<=n;i++) {
34         while(head<=tail&&a[q[tail]]<=a[i]) tail--;
35         q[++tail]=i;
36     }
37     for(int i=k;i<=n;i++) {
38         while(head<=tail&&a[q[tail]]<=a[i]) tail--;
39         q[++tail]=i;
40         while(q[head]<=i-k) head++;
41         cout<<a[q[head]]<<" ";
42     }
43     cout<<endl;
44 }
45
46 int main(){
47     cin>>n>>k; //每k個連續的數
48     for(int i=1;i<=n;i++) cin>>a[i];
49     getmin();
50     getmax();
51     return 0;
52 }

```

5 sort

5.1 大數排序

```

1 #python大數排序
2
3 while True:
4     try:
5         n = int(input())
6         # 有幾筆數字需要排序

```

```

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

```

6 math

6.1 質數與因數

```

1  埃氏篩法
2  int n;
3  vector<int> isprime(n+1,1);
4  isprime[0]=isprime[1]=0;
5  for(int i=2;i*i<=n;i++){
6      if(isprime[i])
7          for(int j=i*i;j<=n;j+=i) isprime[j]=0;
8  }
9
10 歐拉篩O(n)
11 #define MAXN 47000 //sqrt(2^31)=46,340...
12 bool isPrime[MAXN];
13 int prime[MAXN];
14 int primeSize=0;
15 void getPrimes(){
16     memset(isPrime, true, sizeof(isPrime));
17     isPrime[0]=isPrime[1]=false;
18     for(int i=2;i<MAXN;i++){
19         if(isPrime[i]) prime[primeSize++]=i;
20         for(int
21             j=0;j<primeSize&&i*prime[j]<=MAXN;j++){
22             isPrime[i*prime[j]]=false;
23             if(i%prime[j]==0) break;
24         }
25     }
26
27 最大公因數 O(log(min(a,b)))
28 int GCD(int a,int b){
29     if(b==0) return a;
30     return GCD(b,a%b);
31 }
32
33 質因數分解
34 void primeFactorization(int n){
35     for(int i=0;i<(int)p.size();++i){
36         if(p[i]*p[i]>n) break;
37         if(n%p[i]) continue;
38         cout<<p[i]<<' ';
39         while(n%p[i]==0) n/=p[i];
40     }
41     if(n!=1) cout<<n<<' ';
42     cout<<'\\n';
43 }
44
45 擴展歐幾里得算法
46 //ax+by=GCD(a,b)
47 #include <bits/stdc++.h>
48 using namespace std;
49
50 int ext_euc(int a,int b,int &x,int &y){
51     if(b==0){
52         x=1,y=0;
53         return a;
54     }
55     int d=ext_euc(b,a%b,y,x);
56     y-=a/b*x;
57     return d;
58 }
59

```

```

60 int main(){
61     int a,b,x,y;
62     cin>>a>>b;
63     ext_euc(a,b,x,y);
64     cout<<x<<' '<<y<<endl;
65     return 0;
66 }
67
68
69
70 歌德巴赫猜想
71 solution : 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
72 #include <iostream>
73 using namespace std;
74 #define N 2000000
75 int ox[N],p[N],pr;
76 void PrimeTable(){
77     ox[0]=ox[1]=1;
78     pr=0;
79     for(int i=2;i<N;i++){
80         if(!ox[i]) p[pr++]=i;
81         for(int j=0;i*p[j]<N&&j<pr;j++)
82             ox[i*p[j]]=1;
83     }
84 }
85
86 int main(){
87     PrimeTable();
88     int n;
89     while(cin>>n,n){
90         int x;
91         for(x=1;;x+=2)
92             if(!ox[x]&&!ox[n-x]) break;
93         printf("%d = %d + %d\\n",n,x,n-x);
94     }
95 }
96
97 problem : 給定整數 N ,
98 求 N 最少可以拆成多少個質數的和。
99 如果 N 是質數, 則答案為 1。
100 如果 N 是偶數(不包含2), 則答案為 2 (強歌德巴赫猜想)。
101 如果 N 是奇數且 N-2 是質數, 則答案為 2 (2+質數)。
102 其他狀況答案為 3 (弱歌德巴赫猜想)。
103 #include<bits/stdc++.h>
104 using namespace std;
105
106 bool isPrime(int n){
107     for(int i=2;i<n;++i){
108         if(i*i>n) return true;
109         if(n%i==0) return false;
110     }
111     return true;
112 }
113
114 int main(){
115     int n;
116     cin>>n;
117     if(isPrime(n)) cout<<"1\\n";
118     else if(n%2==0||isPrime(n-2)) cout<<"2\\n";
119     else cout<<"3\\n";
120 }

```

6.2 快速幂

```

1  計算a^b
2  #include<iostream>
3  #define ll long long
4  using namespace std;
5
6  const ll MOD=1000000007;
7  ll fp(ll a, ll b) {
8      int ans=1;
9      while(b>0){
10         if(b&1) ans=ans*a%MOD;
11         a=a*a%MOD;
12         b>>=1;

```

```

13     }
14     return ans;
15 }
16
17 int main() {
18     int a,b;
19     cin>>a>>b;
20     cout<<fp(a,b);
21 }

```

6.3 歐拉函數

```

1 //計算閉區間 [1,n] 中的正整數與 n 互質的個數
2
3 int phi(){
4     int ans=n;
5     for(int i=2;i*i<=n;i++){
6         if(n%i==0){
7             ans=ans-ans/i;
8             while(n%i==0) n/=i;
9         }
10    }
11    if(n>1) ans=ans-ans/n;
12    return ans;
13 }

```

6.4 atan

```

1 說明
2  atan() 和 atan2() 函數分別計算 x 和 y/x 的反正切。
3
4 回覆值
5  atan() 函數會傳回介於範圍 - /2 到 /2 弧度之間的值。
6  atan2() 函數會傳回介於 - 至  弧度之間的值。
7  如果 atan2() 函數的兩個引數都是零，
8  則函數會將 errno 設為 EDOM，並傳回值 0。
9
10 範例
11 #include <math.h>
12 #include <stdio.h>
13
14 int main(void){
15     double a,b,c,d;
16
17     c=0.45;
18     d=0.23;
19
20     a=atan(c);
21     b=atan2(c,d);
22
23     printf("atan(%lf)=%lf/n",c,a);
24     printf("atan2(%lf,%lf)=%lf/n",c,d,b);
25
26 }
27
28 /*
29 atan(0.450000)=0.422854
30 atan2(0.450000,0.230000)=1.098299
31 */

```

7 algorithm

7.1 basic

```

1 min_element：找尋最小元素
2 min_element(first, last)
3 max_element：找尋最大元素
4 max_element(first, last)
5 sort：排序，預設由小排到大。
6 sort(first, last)

```

```

7 sort(first, last, cmp)：可自行定義比較運算子 cmp。
8 find：尋找元素。
9 find(first, last, val)
10 lower_bound：尋找第一個小於 x 的元素位置，
11     如果不存在，則回傳 last。
12 lower_bound(first, last, val)
13 upper_bound：尋找第一個大於 x 的元素位置，
14     如果不存在，則回傳 last。
15 upper_bound(first, last, val)
16 next_permutation：將序列順序轉換成下一個字典序，
17     如果存在回傳 true，反之回傳 false。
18 next_permutation(first, last)
19 prev_permutation：將序列順序轉換成上一個字典序，
20     如果存在回傳 true，反之回傳 false。
21 prev_permutation(first, last)

```

7.2 binary search

```

1 int binary_search(int target) {
2     // For range [ok, ng) or (ng, ok], "ok" is for the
3     // index that target value exists, with "ng" doesn't.
4     int ok = maxn, ng = -1;
5     // For first lower_bound, ok=maxn and ng=-1,
6     // for last lower_bound, ok = -1 and ng = maxn
7     // (the "check" funtion
8     // should be changed depending on it.)
9     while(abs(ok - ng) > 1) {
10         int mid = (ok + ng) >> 1;
11         if(check(mid)) ok = mid;
12         else ng = mid;
13     }
14     // Be careful, "arr[mid]>=target" for first
15     // lower_bound and "arr[mid]<=target" for
16     // last lower_bound. For range (ng, ok],
17     // convert it into (ng, mid] and (mid, ok] than
18     // choose the first one, or convert [ok, ng) into
19     // [ok, mid) and [mid, ng) and then choose
20     // the second one.
21     }
22     return ok;
23 }
24
25 lower_bound(arr, arr + n, k); //最左邊 ≥ k 的位置
26 upper_bound(arr, arr + n, k); //最左邊 > k 的位置
27 lower_bound(arr, arr + n, k) - 1; //最右邊 ≤ k 的位置
28 (lower_bound, upper_bound) //等於 k 的範圍
29 equal_range(arr, arr+n, k);

```

7.3 prefix sum

```

1 // 前綴和
2 陣列前n項的和。
3 b[i]=a[0]+a[1]+a[2]+...+a[i]
4 區間和 [l, r]：b[r]-b[l-1] (要保留b[l]所以-1)
5
6 #include<bits/stdc++.h>
7 using namespace std;
8 int main(){
9     int n;
10    cin>>n;
11    int a[n],b[n];
12    for(int i=0;i<n;i++) cin>>a[i];
13    b[0]=a[0];
14    for(int i=1;i<n;i++) b[i]=b[i-1]+a[i];
15    for(int i=0;i<n;i++) cout<<b[i]<<' ';
16    cout<<'\\n';
17    int l,r;
18    cin>>l>>r;
19    cout<<b[r]-b[l-1]; //區間和
20 }

```

7.4 差分

```

1 // 差分
2 用途：在區間 [l, r] 加上一個數字v。
3 b[l] += v; (b[0~l] 加上v)
4 b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v) )
5 給的 a[] 是前綴和數列，建構 b[]，
6 因為 a[i] = b[0] + b[1] + b[2] + ... + b[i]，
7 所以 b[i] = a[i] - a[i-1]。
8 在 b[l] 加上 v，b[r+1] 減去 v，
9 最後再從 0 跑到 n 使 b[i] += b[i-1]。
10 這樣一來，b[] 是一個在某區間加上v的前綴和。
11
12 #include <bits/stdc++.h>
13 using namespace std;
14 int a[1000], b[1000];
15 // a: 前綴和數列, b: 差分數列
16 int main(){
17     int n, l, r, v;
18     cin >> n;
19     for(int i=1; i<=n; i++){
20         cin >> a[i];
21         b[i] = a[i] - a[i-1]; //建構差分數列
22     }
23     cin >> l >> r >> v;
24     b[l] += v;
25     b[r+1] -= v;
26
27     for(int i=1; i<=n; i++){
28         b[i] += b[i-1];
29         cout << b[i] << ' ';
30     }
31 }

```

7.5 greedy

```

1 //貪心
2 貪心演算法的核心為，
3 採取在目前狀態下最好或最佳（即最有利）的選擇。
4 貪心演算法雖然能獲得當前最佳解，
5 但不保證能獲得最後（全域）最佳解，
6 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，
7 確認無誤再實作。
8
9
10 刪數字問題
11 //problem
12 給定一個數字 N( $\leq 10^{100}$ )，需要刪除 K 個數字，
13 請問刪除 K 個數字後最小的數字為何？
14
15 //solution
16 刪除滿足第 i 位數大於第 i+1 位數的最左邊第 i 位數，
17 扣除高位數的影響較扣除低位數的大。
18
19 //code
20 int main(){
21     string s;
22     int k;
23     cin>>s>>k;
24     for(int i=0; i<k; ++i){
25         if((int)s.size()==0) break;
26         int pos = (int)s.size()-1;
27         for(int j=0; j<(int)s.size()-1; ++j){
28             if(s[j]>s[j+1]){
29                 pos=j;
30                 break;
31             }
32         }
33         s.erase(pos, 1);
34     }
35     while((int)s.size()>0&&s[0]=='0')
36         s.erase(0, 1);
37     if((int)s.size()) cout<<s<<'\n';

```

```

38     else cout<<0<<'\n';
39 }
40
41 最小區間覆蓋長度
42 //problem
43 給定 n 條線段區間為 [Li,Ri]，
44 請問最少要選幾個區間才能完全覆蓋 [0,S]?
45
46 //solution
47 先將所有區間依照左界由小到大排序，
48 對於當前區間 [Li,Ri]，要從左界 >Ri 的所有區間中，
49 找到有著最大的右界的區間，連接當前區間。
50
51 //problem
52 長度 n 的直線中有數個加熱器，
53 在 x 的加熱器可以讓 [x-r,x+r] 內的物品加熱，
54 問最少要幾個加熱器可以把 [0,n] 的範圍加熱。
55
56 //solution
57 對於最左邊沒加熱的點a，選擇最遠可以加熱a的加熱器，
58 更新已加熱範圍，重複上述動作繼續尋找加熱器。
59
60 //code
61 int main(){
62     int n, r;
63     int a[1005];
64     cin>>n>>r;
65     for(int i=1; i<=n; ++i) cin>>a[i];
66     int i=1, ans=0;
67     while(i<=n){
68         int R=min(i+r-1, n), L=max(i-r+1, 0)
69         int nextR=-1;
70         for(int j=R; j>=L; --j){
71             if(a[j]){
72                 nextR=j;
73                 break;
74             }
75         }
76         if(nextR==-1){
77             ans=-1;
78             break;
79         }
80         ++ans;
81         i=nextR+r;
82     }
83     cout<<ans<<'\n';
84 }
85
86
87 最多不重疊區間
88 //problem
89 給你 n 條線段區間為 [Li,Ri]，
90 請問最多可以選擇幾條不重疊的線段(頭尾可相連)?
91
92 //solution
93 依照右界由小到大排序，
94 每次取到一個不重疊的線段，答案 +1。
95
96 //code
97 struct Line{
98     int L, R;
99     bool operator<(const Line &rhs) const{
100         return R<rhs.R;
101     }
102 };
103
104 int main(){
105     int t;
106     cin>>t;
107     Line a[30];
108     while(t--){
109         int n=0;
110         while(cin>>a[n].L>>a[n].R, a[n].L||a[n].R)
111             ++n;
112         sort(a, a+n);
113

```



```

114     int ans=1,R=a[0].R;
115     for(int i=1;i<n;i++){
116         if(a[i].L>=R){
117             ++ans;
118             R=a[i].R;
119         }
120     }
121     cout<<ans<<'\n';
122 }
123 }
124
125

```

最小化最大延遲問題

//problem

給定 N 項工作，每項工作的需要處理時長為 T_i ，
期限是 D_i ，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 $\max Li$ 最小。

//solution

按照到期時間從早到晚處理。

//code

```

137 struct Work{
138     int t, d;
139     bool operator< (const Work &rhs) const{
140         return d<rhs.d;
141     }
142 };
143

```

```

144 int main(){
145     int n;
146     Work a[10000];
147     cin>>n;
148     for(int i=0;i<n;++i)
149         cin>>a[i].t>>a[i].d;
150     sort(a,a+n);
151     int maxL=0,sumT=0;
152     for(int i=0;i<n;++i){
153         sumT+=a[i].t;
154         maxL=max(maxL,sumT-a[i].d);
155     }
156     cout<<maxL<<'\n';
157 }
158
159

```

最少延遲數量問題

//problem

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

//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。

//problem

給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？

//solution

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

//code

```

182 struct Work{
183     int t, d;
184     bool operator< (const Work &rhs) const{
185         return d<rhs.d;
186     }
187 };
188

```

```

189 int main(){
190     int n=0;
191     Work a[10000];
192     priority_queue<int> pq;
193     while(cin>>a[n].t>>a[n].d)
194         ++n;
195     sort(a,a+n);
196     int sumT=0,ans=n;
197     for(int i=0;i<n;++i){
198         pq.push(a[i].t);
199         sumT+=a[i].t;
200         if(a[i].d<sumT){
201             int x=pq.top();
202             pq.pop();
203             sumT-=x;
204             --ans;
205         }
206     }
207     cout<<ans<<'\n';
208 }
209

```

任務調度問題

//problem

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

//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 $D_i-T_i+1, D_i-T_i, \dots, 1, 0$ ，
如果有空閒就放進去，否則延後執行。

//problem

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

//solution

和上題相似，這題變成依照獎勵由大到小排序。

//code

```

230 struct Work{
231     int d,p;
232     bool operator< (const Work &rhs) const{
233         return p>rhs.p;
234     }
235 };
236

```

```

237 int main(){
238     int n;
239     Work a[100005];
240     bitset<100005> ok;
241     while(cin>>n){
242         ok.reset();
243         for(int i=0;i<n;++i)
244             cin>>a[i].d>>a[i].p;
245     }
246     sort(a,a+n);
247     int ans=0;
248     for(int i=0;i<n;++i){
249         int j=a[i].d;
250         while(j--){
251             if(!ok[j]){
252                 ans+=a[i].p;
253                 ok[j]=true;
254                 break;
255             }
256         }
257     }
258     cout<<ans<<'\n';
259 }

```

7.6 floyd warshall

```

1 int w[n][n];
2 int d[n][n];
3 int medium[n][n];
4 // 由 i 點到 j 點的路徑，其中繼點為 medium[i][j]。
5
6 void floyd_warshall(){ //O(V^3)
7     for(int i=0;i<n;i++){
8         for(int j=0;j<n;j++){
9             d[i][j]=w[i][j];
10            medium[i][j]=-1;
11            // 預設為沒有中繼點
12        }
13        for(int i=0;i<n;i++) d[i][i]=0;
14        for(int k=0;k<n;k++){
15            for(int i=0;i<n;i++){
16                for(int j=0;j<n;j++){
17                    if(d[i][k]+d[k][j]<d[i][j]){
18                        d[i][j]=d[i][k]+d[k][j];
19                        medium[i][j]=k;
20                        // 由 i 點走到 j 點經過了 k 點
21                    }
22                }
23            }
24            // 這支函式並不會印出起點和終點，必須另行印出。
25        }
26        void find_path(int s,int t){ // 印出最短路徑
27            if(medium[s][t]==-1) return; // 沒有中繼點就結束
28            find_path(s,medium[s][t]); // 前半段最短路徑
29            cout<<medium[s][t]; // 中繼點
30            find_path(medium[s][t],t); // 後半段最短路徑
31        }
32    }
33 }

```

7.7 dinic

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <queue>
4 #define MAXNODE 105
5 #define oo 1e9
6 using namespace std;
7
8 int nodeNum;
9 int graph[MAXNODE][MAXNODE];
10 int levelGraph[MAXNODE];
11 bool canReachSink[MAXNODE];
12
13 bool bfs(int from, int to){
14     memset(levelGraph,0,sizeof(levelGraph));
15     levelGraph[from]=1;
16     queue<int> q;
17     q.push(from);
18     int currentNode;
19     while(!q.empty()){
20         currentNode=q.front();
21         q.pop();
22         for(int nextNode=1;nextNode<=nodeNum;
23             ++nextNode){
24             if((levelGraph[nextNode]==0)&&
25                 graph[currentNode][nextNode]>0){
26                 levelGraph[nextNode]=
27                     levelGraph[currentNode]+1;
28                 q.push(nextNode);
29             }
30             if((nextNode==to)&&
31                 (graph[currentNode][nextNode]>0))
32                 return true;
33         }
34     }
35     return false;
36 }
37
38 int dfs(int from, int to, int bottleNeck){
39     if(from == to) return bottleNeck;
40     int outFlow = 0;
41     int flow;
42     for(int nextNode=1;nextNode<=nodeNum;++nextNode){
43         if((graph[from][nextNode]>0)&&

```

```

43         (levelGraph[nextNode]==levelGraph[from]+1)&&
44         canReachSink[nextNode]){
45             flow=dfs(nextNode,to,
46                 min(graph[from][nextNode],bottleNeck));
47             graph[from][nextNode]-=flow; //貪心
48             graph[nextNode][from]+=flow; //反悔路
49             outFlow+=flow;
50             bottleNeck-=flow;
51         }
52         if(bottleNeck==0) break;
53     }
54     if(outFlow==0) canReachSink[from]=false;
55     return outFlow;
56 }
57
58 int dinic(int from, int to){
59     int maxFlow=0;
60     while(bfs(from, to)){
61         memset(canReachSink,1,sizeof(canReachSink));
62         maxFlow += dfs(from, to, oo);
63     }
64     return maxFlow;
65 }
66
67 int main(){
68     int from, to, edgeNum;
69     int NetWorkNum = 1;
70     int maxFlow;
71     while(scanf("%d",&nodeNum)!=EOF&&nodeNum!=0){
72         memset(graph, 0, sizeof(graph));
73         scanf("%d %d %d", &from, &to, &edgeNum);
74         int u, v, w;
75         for (int i = 0; i < edgeNum; ++i){
76             scanf("%d %d %d", &u, &v, &w);
77             graph[u][v] += w;
78             graph[v][u] += w;
79         }
80         maxFlow = dinic(from, to);
81         printf("Network %d\n", NetWorkNum++);
82         printf("The bandwidth is %d.\n\n", maxFlow);
83     }
84     return 0;
85 }

```

7.8 SegmentTree

```

1 #define MAXN 1000
2 int data[MAXN]; //原數據
3 int st[4 * MAXN]; //線段樹
4 int tag[4 * MAXN]; //懶標
5
6 inline int pull(int l, int r) {
7     // 隨題目改變 sum、max、min
8     // l、r 是左右樹的 index
9     return st[l] + st[r];
10 }
11
12 void build(int l, int r, int i) {
13     // 在 [l, r] 區間建樹，目前根的 index 為 i
14     if (l == r) {
15         st[i] = data[l];
16         return;
17     }
18     int mid = 1 + ((r - l) >> 1);
19     build(l, mid, i * 2);
20     build(mid + 1, r, i * 2 + 1);
21     st[i] = pull(i * 2, i * 2 + 1);
22 }
23
24 int query(int ql, int qr, int l, int r, int i) {
25     // [ql, qr] 是查詢區間，[l, r] 是當前節點包含的區間
26     if (ql <= l && r <= qr)
27         return st[i];
28     int mid = 1 + ((r - l) >> 1);
29     if (tag[i]) {

```



```

30 //如果當前懶標有值則更新左右節點
31 st[i * 2] += tag[i] * (mid - 1 + 1);
32 st[i * 2 + 1] += tag[i] * (r - mid);
33 tag[i * 2] += tag[i]; //下傳懶標至左節點
34 tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
35 tag[i] = 0;
36 }
37 int sum = 0;
38 if (ql <= mid)
39     sum += query(ql, qr, 1, mid, i * 2);
40 if (qr > mid)
41     sum += query(ql, qr, mid + 1, r, i * 2 + 1);
42 return sum;
43 }
44 void update(int ql, int qr, int l, int r, int i, int c) {
45 // [ql, qr]是查詢區間, [l, r]是當前節點包含的區間
46 // c是變化量
47 if (ql <= l && r <= qr) {
48     st[i] += (r - l + 1) * c;
49     //求和, 此需乘上區間長度
50     tag[i] += c;
51     return;
52 }
53 int mid = l + ((r - l) >> 1);
54 if (tag[i] && l != r) {
55     //如果當前懶標有值則更新左右節點
56     st[i * 2] += tag[i] * (mid - l + 1);
57     st[i * 2 + 1] += tag[i] * (r - mid);
58     tag[i * 2] += tag[i]; //下傳懶標至左節點
59     tag[i * 2 + 1] += tag[i]; //下傳懶標至右節點
60     tag[i] = 0;
61 }
62 if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
63 if (qr > mid) update(ql, qr, mid + 1, r, i * 2 + 1, c);
64 st[i] = pull(i * 2, i * 2 + 1);
65 }
66 //如果是直接改值而不是加值, query與update中的tag與st的
67 //改值從+=改成=

```

7.9 Nim Game

```

1 //兩人輪流取銅板, 每人每次需在某堆取一枚以上的銅板,
2 //但不能同時在兩堆取銅板, 直到最後,
3 //將銅板拿光的人贏得此遊戲。
4
5 #include <bits/stdc++.h>
6 #define maxn 23+5
7 using namespace std;
8
9 int SG[maxn];
10 int visited[1000+5];
11 int pile[maxn], ans;
12
13 void calculateSG() {
14     SG[0] = 0;
15     for (int i = 1; i <= maxn; i++) {
16         int cur = 0;
17         for (int j = 0; j < i; j++)
18             for (int k = 0; k <= j; k++)
19                 visited[SG[j]^SG[k]] = i;
20         while (visited[cur] == i) cur++;
21         SG[i] = cur;
22     }
23 }
24
25 int main() {
26     calculateSG();
27     int Case = 0, n;
28     while (cin >> n, n) {
29         ans = 0;
30         for (int i = 1; i <= n; i++) cin >> pile[i];
31         for (int i = 1; i <= n; i++) if (pile[i] & 1)
32             ans ^= SG[n - i];

```

```

32     cout << "Game " << ++Case << ": ";
33     if (!ans) cout << "-1 -1 -1\n";
34     else {
35         bool flag = 0;
36         for (int i = 1; i <= n; i++) {
37             if (pile[i]) {
38                 for (int j = i + 1; j <= n; j++) {
39                     for (int k = j; k <= n; k++) {
40                         if ((SG[n - i]^SG[n - j]^SG[n - k]) == ans) {
41                             cout << i - 1 << " " << j - 1 << " " << k - 1 << endl;
42                             flag = 1;
43                             break;
44                         }
45                     }
46                     if (flag) break;
47                 }
48                 if (flag) break;
49             }
50         }
51     }
52     return 0;
53 }
54
55 /*
56 input
57 4 1 0 1 100
58 3 1 0 5
59 2 2 1
60 0
61 output
62 Game 1: 0 2 3
63 Game 2: 0 1 1
64 Game 3: -1 -1 -1
65 */

```

7.10 Trie

```

1 #include <bits/stdc++.h>
2 #define word_maxn 4000*100+5
3 #define str_maxn 300000+5
4 #define sigma_num 26
5 #define MOD 20071027
6 using namespace std;
7
8 int dp[str_maxn];
9 char S[str_maxn];
10 char wd[100+5];
11
12 struct Trie {
13     int ch[word_maxn][sigma_num];
14     int val[word_maxn];
15     int seq;
16     void init() {
17         seq = 1;
18         memset(ch, 0, sizeof(ch));
19     }
20     void insertion(char *s) {
21         int row = 0, n = strlen(s);
22         for (int i = 0; i < n; i++) {
23             int letter_no = s[i] - 'a';
24             if (ch[row][letter_no] == 0) {
25                 ch[row][letter_no] = seq;
26                 memset(ch[seq], 0, sizeof(ch[seq]));
27                 val[seq++] = 0;
28             }
29             row = ch[row][letter_no];
30         }
31         val[row] = n;
32     }
33     void find_prefix(char *s, int len, vector<int> &vc) {
34         int row = 0;
35         for (int i = 0; i < len; i++) {
36             int letter_no = s[i] - 'a';
37             if (ch[row][letter_no] == 0) return;
38             row = ch[row][letter_no];

```

```

39         if(val[row]) vc.push_back(val[row]);
40     }
41 }
42 }tr;
43
44 int main(){
45     int Case=1;
46     while(cin>>S){
47         int n;
48         cin>>n;
49         tr.init();
50         for(int i=0;i<n;i++){
51             cin>>wd;
52             tr.insertion(wd);
53         }
54         memset(dp,0,sizeof(dp));
55         int N=strlen(S);
56         dp[N]=1;
57         for(int i=N-1;i>=0;i--){
58             vector<int> vc;
59             tr.find_prefix(S+i,N-i,vc);
60             for(int j=0;j<vc.size();j++){
61                 dp[i]=(dp[i]+dp[i+vc[j]])%MOD;
62             }
63             cout<<"Case " <<Case++<<" : "<<dp[0]<<endl;
64         }
65         return 0;
66     }
67     /*
68     input
69     abcd
70     4
71     a b cd ab
72     output
73     Case 1: 2
74     */

```

7.11 SPFA

```

1 struct Edge
2 {
3     int t;
4     long long w;
5     Edge(){};
6     Edge(int _t, long long _w) : t(_t), w(_w) {}
7 };
8
9 bool SPFA(int st) // 平均 $O(V + E)$  最糟 $O(VE)$ 
10 {
11     vector<int> cnt(n, 0);
12     bitset<MXV> inq(0);
13     queue<int> q;
14     q.push(st);
15     dis[st] = 0;
16     inq[st] = true;
17     while (!q.empty())
18     {
19         int cur = q.front();
20         q.pop();
21         inq[cur] = false;
22         for (auto &e : G[cur])
23         {
24             if (dis[e.t] <= dis[cur] + e.w)
25                 continue;
26             dis[e.t] = dis[cur] + e.w;
27             if (inq[e.t])
28                 continue;
29             ++cnt[e.t];
30             if (cnt[e.t] > n)
31                 return false; // negative cycle
32             inq[e.t] = true;
33             q.push(e.t);
34         }
35     }

```

```

36     return true;
37 }

```

7.12 dijkstra

```

1 #include<bits/stdc++.h>
2 #define maxn 50000+5
3 #define INF 0x3f3f3f3f
4 using namespace std;
5
6 struct edge{
7     int v,w;
8 };
9
10 struct Item{
11     int u,dis;
12     bool operator< (const Item &rhs) const{
13         return dis>rhs.dis;
14     }
15 };
16
17 vector<edge> G[maxn];
18 int dist[maxn];
19
20 void dijkstra(int s){ //  $O((V + E)\log E)$ 
21     memset(dist,INF,sizeof(dist));
22     dist[s]=0;
23     priority_queue<Item> pq;
24     pq.push({s,0});
25     while(!pq.empty()){
26         Item now=pq.top();
27         pq.pop();
28         if(now.dis>dist[now.u]) continue;
29         for(edge e:G[now.u]){
30             if(dist[e.v]>dist[now.u]+e.w){
31                 dist[e.v]=dist[now.u]+e.w;
32                 pq.push({e.v,dist[e.v]});
33             }
34         }
35     }
36 }
37
38 int main(){
39     int t,cas=1;
40     cin>>t;
41     while(t--){
42         int n,m,s,t;
43         cin>>n>>m>>s>>t;
44         for(int i=0;i<n;i++) G[i].clear();
45         int u,v,w;
46         for(int i=0;i<m;i++){
47             cin>>u>>v>>w;
48             G[u].push_back({v,w});
49             G[v].push_back({u,w});
50         }
51         dijkstra(s);
52         cout<<"Case #" <<cas++<<" : ";
53         if(dist[t]==INF) cout<<"unreachable\n";
54         else cout<<dist[t]<<endl;
55     }
56 }

```

8 動態規劃

8.1 LCS 和 LIS

```

1 //最長共同子序列 (LCS)
2 給定兩序列 A,B , 求最長的序列 C ,
3 C 同時為 A,B 的子序列。
4
5 //最長遞增子序列 (LIS)
6 給你一個序列 A , 求最長的序列 B ,

```

```

7 | B 是一個（非）嚴格遞增序列，且為 A 的子序列。
8 |
9 | //LCS 和 LIS 題目轉換
10 | LIS 轉成 LCS
11 |     1. A 為原序列， B=sort(A)
12 |     2. 對 A,B 做 LCS
13 | LCS 轉成 LIS
14 |     1. A, B 為原本的兩序列
15 |     2. 最 A 序列作編號轉換，將轉換規則套用在 B
16 |     3. 對 B 做 LIS
17 |     4. 重複的數字在編號轉換時後要變成不同的數字，
18 |         越早出現的數字要越小
19 |     5. 如果有數字在 B 裡面而不在 A 裡面，
20 |         直接忽略這個數字不做轉換即可

```

9 Section2

9.1 thm

- 中文測試

- $$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$