

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	4
5.1	大數排序	4
6	math	4
6.1	質數與因數	4
6.2	prime factorization	5
6.3	快速冪	5
6.4	歐拉函數	5
7	algorithm	5
7.1	basic	5
7.2	binarysearch	5
7.3	prefix sum	6
7.4	差分	6
7.5	greedy	6
7.6	floydwarshall	9
7.7	dinic	9
8	動態規劃	10
8.1	LCS 和 LIS	10
9	Section2	10
9.1	thm	10

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

1	int	char	int	char	int	char
2	32		64	@	96	`
3	33	!	65	A	97	a
4	34	"	66	B	98	b
5	35	#	67	C	99	c
6	36	\$	68	D	100	d
7	37	%	69	E	101	e

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

```

22 | r[0] = 1;
23 | for(int i = 1; i < n; i++) {
24 |     int ii = center - (i - center);
25 |     int len = mx - i + 1;
26 |     if(i > mx) {
27 |         r[i] = ex(i, i);
28 |         center = i;
29 |         mx = i + r[i] - 1;
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 |     } else {
35 |         r[i] = min(r[ii], len);
36 |     }
37 |     ans = max(ans, r[i]);
38 | }
39 |
40 | cout << ans - 1 << "\n";
41 | return 0;
42 | }

```

## 3.2 stringstream

```

1 | string s, word;
2 | stringstream ss;
3 | getline(cin, s);
4 | ss << s;
5 | while(ss >> word)
6 |     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，
8 | 讓我們不必煩惱佇列長度的問題。
9 | dq.push_back() //在 deque 的最尾端新增元素
10 | dq.push_front() //在 deque 的開頭新增元素
11 | dq.pop_back() //移除 deque 最尾端的元素
12 | dq.pop_front() //移除 deque 最開頭的元素
13 | dq.back() //取出 deque 最尾端的元素
14 | dq.front() //回傳 deque 最開頭的元素
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>("r000",
42 |         "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 << " " << iter->second << endl;
49 |     }
50 |     for(iter_r = mp.rbegin(); iter_r != mp.rend();
51 |         iter_r++){
52 |         cout << iter_r->first << "
53 |             " << iter_r->second << endl;
54 |     }
55 |     //find and erase the element

```

```

52 |     iter = mp.find("r123");
53 |     mp.erase(iter);
54 |     iter = mp.find("r123");
55 |     if(iter != mp.end())
56 |         cout<<"Find, the value is
           "<<iter->second<<endl;
57 |     else
58 |         cout<<"Do not Find"<<endl;
59 |     return 0;
60 | }

```

## 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 初始化  
 6 unordered\_set<int> myunordered\_set{1, 2, 3, 4, 5};  
 7  
 8 陣列初始化  
 9 int arr[] = {1, 2, 3, 4, 5};  
 10 unordered\_set<int> myunordered\_set(arr, arr+5);  
 11  
 12 插入元素  
 13

```

14 unordered_set<int> myunordered_set;
15 myunordered_set.insert(1);
16
17 迴圈遍歷 unordered_set 容器
18 #include <iostream>
19 #include <unordered_set>
20 using namespace std;
21 int main() {
22     unordered_set<int> myunordered_set = {3, 1};
23     myunordered_set.insert(2);
24     myunordered_set.insert(5);
25     myunordered_set.insert(4);
26     myunordered_set.insert(5);
27     myunordered_set.insert(4);
28     for (const auto &s : myunordered_set)
29         cout << s << " ";
30     cout << "\n";
31     return 0;
32 }
33
34 /*
35 output
36 4 5 2 1 3
37 */
38
39 unordered_set 刪除指定元素
40 #include <iostream>
41 #include <unordered_set>
42 int main() {
43     unordered_set<int> myunordered_set{2, 4, 6, 8};
44     myunordered_set.erase(2);
45     for (const auto &s : myunordered_set)
46         cout << s << " ";
47     cout << "\n";
48     return 0;
49 }
50 /*
51 output
52 8 6 4
53 */
54
55 清空 unordered_set 元素
56 unordered_set<int> myunordered_set;
57 myunordered_set.insert(1);
58 myunordered_set.clear();
59
60 unordered_set 判斷元素是否存在
61 unordered_set<int> myunordered_set;
62 myunordered_set.insert(2);
63 myunordered_set.insert(4);
64 myunordered_set.insert(6);
65 cout << myunordered_set.count(4) << "\n"; // 1
66 cout << myunordered_set.count(8) << "\n"; // 0
67
68 判斷 unordered_set 容器是否為空
69 #include <iostream>
70 #include <unordered_set>
71
72 int main() {
73     unordered_set<int> myunordered_set;
74     myunordered_set.clear();
75     if(myunordered_set.empty())
76         cout<<"empty\n";
77     else
78         cout<<"not empty, size is
           "<<myunordered_set.size()<<"\n";
79     return 0;
80 }

```

## 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<k; 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<k; 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())           # 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 | 質數
2 |
3 | 埃氏篩法
4 | int n;
5 | vector<int> isprime(n+1,1);
6 | isprime[0]=isprime[1]=0;
7 | for(int i=2; i*i<=n; i++){
8 |     if(isprime[i])
9 |         for(int j=i*i; j<=n; j+=i) isprime[j]=0;
10 | }
11 |
12 | 歐拉篩O(n)
13 | #define MAXN 47000 // sqrt(2^31) = 46,340...
14 | bool isPrime[MAXN];
15 | int prime[MAXN];
16 | int primeSize = 0;
17 | void getPrimes(){
18 |     memset(isPrime, true, sizeof(isPrime));
19 |     isPrime[0] = isPrime[1] = false;
20 |     for (int i = 2; i < MAXN; i++){
21 |         if (isPrime[i]) prime[primeSize++] = i;
22 |         for (int j = 0; j < primeSize && i * prime[j]
23 |             <= MAXN; ++j){
24 |             isPrime[i * prime[j]] = false;
25 |             if (i % prime[j] == 0) break;
26 |         }
27 |     }
28 | }
29 |
30 | 因數
31 |
32 | 最大公因數 O(log(min(a,b)))
33 | int GCD(int a, int b)
34 | {
35 |     if (b == 0) return a;
36 |     return GCD(b, a % b);
37 | }
38 |
39 | 質因數分解
40 |
41 | void primeFactorization(int n){
42 |     for(int i=0; i<(int)p.size(); ++i){
43 |         if(p[i] * p[i] > n) break;
44 |         if(n % p[i]) continue;
45 |         cout << p[i] << ' ';
46 |         while(n % p[i] == 0) n /= p[i];
47 |     }
48 |     if(n!=1) cout<<n<<' ';
49 |     cout<<'\n';
50 | }
51 |
52 | 歌德巴赫猜想
53 | solution : 把偶數 N (6≤N≤10^6) 寫成兩個質數的和。
54 | #include <iostream>
55 | #include <cstdio>
56 | using namespace std;
57 | #define N 20000000
58 | int ox[N], p[N], pr;
59 | void PrimeTable(){
60 |     ox[0] = ox[1] = 1;
61 |     pr = 0;
62 |     for (int i = 2; i < N; i++){
63 |         if (!ox[i]) p[pr++] = i;
64 |         for (int j = 0; i*p[j]<N&&j < pr; j++)
65 |             ox[i*p[j]] = 1;
66 |     }
67 | }
68 |
69 | int main(){
70 |     PrimeTable();
71 |     int n;
72 |     while (cin>>n,n){
73 |         int x;
74 |         for (x = 1;; x += 2)
75 |             if (!ox[x] && !ox[n - x])break;
76 |         printf("%d = %d + %d\n", n, x, n - x);

```

```

77     }
78 }
79 problem : 給定整數 N，求 N
           最少可以拆成多少個質數的和。
80 如果 N 是質數，則答案為 1。
81 如果 N 是偶數(不包含2)，則答案為 2 (強歌德巴赫猜想)。
82 如果 N 是奇數且 N-2 是質數，則答案為 2 (2+質數)。
83 其他狀況答案為 3 (弱歌德巴赫猜想)。
84 #pragma GCC optimize("O2")
85 #include <bits/stdc++.h>
86 using namespace std;
87 #define FOR(i, L, R) for(int i=L;i<(int)R;++i)
88 #define FORD(i, L, R) for(int i=L;i>(int)R;--i)
89 #define IOS
90     cin.tie(nullptr);
91     cout.tie(nullptr);
92     ios_base::sync_with_stdio(false);
93
94 bool isPrime(int n){
95     FOR(i, 2, n){
96         if (i * i > n)
97             return true;
98         if (n % i == 0)
99             return false;
100     }
101     return true;
102 }
103
104 int main(){
105     IOS;
106     int n;
107     cin >> n;
108     if(isPrime(n)) cout << "1\n";
109     else if(n%2==0||isPrime(n-2)) cout<<"2\n";
110     else cout << "3\n";
111 }

```

## 6.2 prime factorization

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     while(true) {
7         cin>>n;
8         for(int x=2; x<=n; x++) {
9             while(n%x==0) {
10                 cout<<x<<"*";
11                 n/=x;
12             }
13         }
14         cout<<"\b \n";
15     }
16     system("pause");
17     return 0;
18 }

```

## 6.3 快速冪

```

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 }

```

## 6.4 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 #include <bits/stdc++.h>
3 using namespace std;
4 int n,ans;
5
6 int phi(){
7     ans=n;
8     for(int i=2;i*i<=n;i++){
9         if(n%i==0){
10             ans=ans-ans/i;
11             while(n%i==0) n/=i;
12         }
13     }
14     if(n>1) ans=ans-ans/n;
15     return ans;
16 }
17
18 int main(){
19     while(cin>>n)
20         cout<<phi()<<endl;

```

## 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 binarysearch

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int binary_search(vector<int> &nums, int target) {
5     int left=0, right=nums.size()-1;
6     while(left<=right){
7         int mid=(left+right)/2;
8         if (nums[mid]>target) right=mid-1;
9         else if(nums[mid]<target) left=mid+1;

```

```

10     else return mid+1;
11 }
12 return 0;
13 }
14
15 int main() {
16     int n, k, x;
17     cin >> n >> k;
18     int a[n];
19     vector<int> v;
20     for(int i=0 ; i<n ; i++){
21         cin >> x;
22         v.push_back(x);
23     }
24     for(int i=0 ; i<k ; i++) cin >> a[i];
25     for(int i=0 ; i<k ; i++){
26         cout << binary_search(v, a[i]) << endl;
27     }
28 }
29
30 lower_bound(a, a + n, k);    //最左邊 ≥ k 的位置
31 upper_bound(a, a + n, k);    //最左邊 > k 的位置
32 upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
33 lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
34 (lower_bound, upper_bound)   //等於 k 的範圍
35 equal_range(a, a+n, k);
36
37 /*
38 input
39 5 5
40 1 3 4 7 9
41 3 1 9 7 -2
42 */
43
44 /*
45 output
46 2
47 1
48 5
49 4
50 0
51 */

```

### 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 霍夫曼樹的變形題
11 //problem
12 給定 N 個數，每次將兩個數 a,b 合併成 a+b，
13 只到最後只剩一個數，合併成本為兩數和，
14 問最小合併成本為多少。
15
16 //solution
17 每次將最小的兩數合併起來。
18
19 //code
20 #include <bits/stdc++.h>
21 using namespace std;
22 int main()
23 {
24     int n, x;
25     while (cin >> n, n){
26         priority_queue<int, vector<int>, greater<int>>
27             q;
28         while (n--){
29             cin >> x;
30             q.push(x);
31         }
32         long long ans = 0;
33         while (q.size() > 1){
34             x = q.top();
35             q.pop();
36             x += q.top();
37             q.pop();
38             q.push(x);
39             ans += x;
40         }
41         cout << ans << endl;
42     }
43 }

```

```

44 刪數字問題
45 //problem
46 給定一個數字  $N(\leq 10^{100})$ ，需要刪除  $K$  個數字，
47 請問刪除  $K$  個數字後最小的數字為何？
48
49 //solution
50 刪除滿足第  $i$  位數大於第  $i+1$  位數的最左邊第  $i$  位數，
51 扣除高位數的影響較扣除低位數的大。
52
53 //code
54 int main()
55 {
56     string s;
57     int k;
58     cin >> s >> k;
59     for (int i = 0; i < k; ++i){
60         if ((int)s.size() == 0) break;
61         int pos = (int)s.size() - 1;
62         for (int j = 0; j < (int)s.size() - 1; ++j){
63             if (s[j] > s[j + 1]){
64                 pos = j;
65                 break;
66             }
67         }
68         s.erase(pos, 1);
69     }
70     while ((int)s.size() > 0 && s[0] == '0')
71         s.erase(0, 1);
72     if ((int)s.size()) cout << s << '\n';
73     else cout << 0 << '\n';
74 }
75
76 區間覆蓋長度
77 //problem
78 給定  $n$  條線段區間為  $[Li, Ri]$ ，
79 請問這些線段的覆蓋所覆蓋的長度？
80
81 //solution
82 先將所有區間依照左界由小到大排序，
83 左界相同依照右界由小到大排序，
84 用一個變數  $R$  紀錄目前最大可以覆蓋到的右界。
85 如果目前區間左界  $\leq R$ ，代表該區間可以和前面的線段合併。
86
87 //code
88 struct Line
89 {
90     int L, R;
91     bool operator<(const Line &rhs) const
92     {
93         if (L != rhs.L) return L < rhs.L;
94         return R < rhs.R;
95     }
96 };
97
98 int main(){
99     int n;
100     Line a[10005];
101     while (cin >> n){
102         for (int i = 0; i < n; i++){
103             cin >> a[i].L >> a[i].R;
104             sort(a, a + n);
105             int ans = 0, L = a[0].L, R = a[0].R;
106             for (int i = 1; i < n; i++){
107                 if (a[i].L < R) R = max(R, a[i].R);
108                 else{
109                     ans += R - L;
110                     L = a[i].L;
111                     R = a[i].R;
112                 }
113             }
114             cout << ans + (R - L) << '\n';
115         }
116     }
117 }
118
119

```

```

120 最小區間覆蓋長度
121 //problem
122 給定  $n$  條線段區間為  $[Li, Ri]$ ，
123 請問最少要選幾個區間才能完全覆蓋  $[0, S]$ ？
124
125 //solution
126 先將所有區間依照左界由小到大排序，
127 對於當前區間  $[Li, Ri]$ ，要從左界  $> Ri$  的所有區間中，
128 找到有著最大的右界的區間，連接當前區間。
129
130 //problem
131 長度  $n$  的直線中有數個加熱器，
132 在  $x$  的加熱器可以讓  $[x-r, x+r]$  內的物品加熱，
133 問最少要幾個加熱器可以把  $[0, n]$  的範圍加熱。
134
135 //solution
136 對於最左邊沒加熱的點  $a$ ，選擇最遠可以加熱  $a$  的加熱器，
137 更新已加熱範圍，重複上述動作繼續尋找加熱器。
138
139 //code
140 int main(){
141     int n, r;
142     int a[1005];
143     cin >> n >> r;
144     for (int i = 1; i <= n; ++i) cin >> a[i];
145     int i = 1, ans = 0;
146     while (i <= n){
147         int R = min(i + r - 1, n), L = max(i - r + 1, 0);
148         int nextR = -1;
149         for (int j = R; j >= L; --j){
150             if (a[j]){
151                 nextR = j;
152                 break;
153             }
154         }
155         if (nextR == -1){
156             ans = -1;
157             break;
158         }
159         ++ans;
160         i = nextR + r;
161     }
162     cout << ans << '\n';
163 }
164
165 最多不重疊區間
166 //problem
167 給你  $n$  條線段區間為  $[Li, Ri]$ ，
168 請問最多可以選擇幾條不重疊的線段(頭尾可相連)？
169
170 //solution
171 依照右界由小到大排序，
172 每次取到一個不重疊的線段，答案  $+1$ 。
173
174 //code
175 struct Line
176 {
177     int L, R;
178     bool operator<(const Line &rhs) const {
179         return R < rhs.R;
180     }
181 };
182
183 int main(){
184     int t;
185     cin >> t;
186     Line a[30];
187     while (t--){
188         int n = 0;
189         while (cin >> a[n].L >> a[n].R, a[n].L || a[n].R)
190             ++n;
191         sort(a, a + n);
192         int ans = 1, R = a[0].R;
193         for (int i = 1; i < n; i++){
194             if (a[i].L >= R){
195

```



```

196         ++ans;
197         R = a[i].R;
198     }
199 }
200 cout << ans << '\n';
201 }
202 }
203
204 區間選點問題
205 //problem
206 給你 n 條線段區間為 [Li,Ri]，
207 請問至少要取幾個點才能讓每個區間至少包含一個點？
208
209 //solution
210 將區間依照右界由小到大排序，R=第一個區間的右界，
211 遍歷所有區段，如果當前區間左界>R，
212 代表必須多選一個點 (ans+=1)，並將 R=當前區間右界。
213
214 //problem
215 給定 N 個座標，要在 x 軸找到最小的點，
216 讓每個座標至少和一個點距離 ≤ D。
217
218 //solution
219 以每個點 (xi,yi) 為圓心半徑為 D 的圓 C，
220 求出 C 和 x 軸的交點 Li,Ri，題目轉變成區間選點問題。
221
222 //code
223 struct Line
224 {
225     int L, R;
226     bool operator<(const Line &rhs) const {
227         return R < rhs.R;
228     }
229 };
230
231 int main(){
232     int t;
233     cin >> t;
234     Line a[30];
235     while (t--){
236         int n = 0;
237         while (cin>>a[n].L>>a[n].R, a[n].L||a[n].R)
238             ++n;
239         sort(a, a + n);
240         int ans = 1, R = a[0].R;
241         for (int i = 1; i < n; i++){
242             if (a[i].L >= R){
243                 ++ans;
244                 R = a[i].R;
245             }
246         }
247         cout << ans << '\n';
248     }
249 }
250
251
252 最小化最大延遲問題
253 //problem
254 給定 N 項工作，每項工作的需要處理時長為 Ti，
255 期限是 Di，第 i 項工作延遲的時間為 Li=max(0,Fi-Di)，
256 原本Fi 為第 i 項工作的完成時間，
257 求一種工作排序使 maxLi 最小。
258
259 //solution
260 按照到期時間從早到晚處理。
261
262 //code
263 struct Work
264 {
265     int t, d;
266     bool operator<(const Work &rhs) const {
267         return d < rhs.d;
268     }
269 };
270
271

```

```

272 int main(){
273     int n;
274     Work a[10000];
275     cin >> n;
276     for (int i = 0; i < n; ++i)
277         cin >> a[i].t >> a[i].d;
278     sort(a, a + n);
279     int maxL = 0, sumT = 0;
280     for (int i = 0; i < n; ++i){
281         sumT += a[i].t;
282         maxL = max(maxL, sumT - a[i].d);
283     }
284     cout << maxL << '\n';
285 }
286
287 最少延遲數量問題
288 //problem
289 給定 N 個工作，每個工作的需要處理時長為 Ti，
290 期限是 Di，求一種工作排序使得逾期工作數量最小。
291
292 //solution
293 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，
294 依序放入工作列表中，如果發現有工作預期，
295 就從目前選擇的工作中，移除耗時最長的工作。
296
297 上述方法為 Moore-Hodgson s Algorithm。
298
299 //problem
300 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
301
302 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
303
304 工作處理時長 → 烏龜重量
305 工作期限 → 烏龜可承受重量
306 多少工作不延期 → 可以疊幾隻烏龜
307
308 //code
309 struct Work{
310     int t, d;
311     bool operator<(const Work &rhs) const {
312         return d < rhs.d;
313     }
314 };
315
316 int main(){
317     int n = 0;
318     Work a[10000];
319     priority_queue<int> pq;
320     while(cin >> a[n].t >> a[n].d)
321         ++n;
322     sort(a, a + n);
323     int sumT = 0, ans = n;
324     for (int i = 0; i < n; ++i){
325         pq.push(a[i].t);
326         sumT += a[i].t;
327         if(a[i].d<sumT){
328             int x = pq.top();
329             pq.pop();
330             sumT -= x;
331             --ans;
332         }
333     }
334     cout << ans << '\n';
335 }
336
337 任務調度問題
338 //problem
339 給定 N 項工作，每項工作的需要處理時長為 Ti，
340 期限是 Di，如果第 i 項工作延遲需要受到 pi 單位懲罰，
341 請問最少會受到多少單位懲罰。
342
343 //solution
344 依照懲罰由大到小排序，
345 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。

//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

```
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';
    }
}
```

多機調度問題

//problem

給定  $N$  項工作，每項工作的需要處理時長為  $T_i$ ，  
有  $M$  台機器可執行多項工作，但不能將工作拆分，  
最快可以在什麼時候完成所有工作？

//solution

將工作由大到小排序，每項工作交給最快空間的機器。

//code

```
int main(){
    int n, m;
    int a[10000];
    cin >> n >> m;
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    sort(a, a + n, greater<int>());
    int ans = 0;
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 0; i < m && i < n; ++i){
        ans = max(ans, a[i]);
        pq.push(a[i]);
    }
    for (int i = m; i < n; ++i){
        int x = pq.top();
        pq.pop();
        x += a[i];
        ans = max(ans, x);
        pq.push(x);
    }
    cout << ans << '\n';
}
```

## 7.6 floydwarshall

```
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(){
    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
}
```

## 7.7 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;
```

## 9 Section2

### 9.1 thm

• 中文測試

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

```

40 int flow;
41 for(int nextNode=1;nextNode<=nodeNum;++nextNode){
42     if((graph[from][nextNode]>0)&&
43         (levelGraph[from]==levelGraph[nextNode]-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 }

```

## 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

```