

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
4	STL	2
4.1	priority_queue . . . . .	2
4.2	queue . . . . .	2
4.3	deque . . . . .	2
4.4	map . . . . .	2
4.5	unordered_map . . . . .	3
4.6	set . . . . .	3
4.7	multiset . . . . .	3
4.8	unordered_set . . . . .	3
4.9	單調隊列 . . . . .	4
5	sort	5
5.1	大數排序 . . . . .	5
5.2	bubble sort . . . . .	5
6	math	5
6.1	質數與因數 . . . . .	5
6.2	prime factorization . . . . .	6
6.3	快速冪 . . . . .	6
7	algorithm	7
7.1	basic . . . . .	7
7.2	binarysearch . . . . .	7
7.3	prefix sum . . . . .	7
7.4	差分 . . . . .	7
7.5	greedy . . . . .	8

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
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]
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 }

```

## 4 STL

### 4.1 priority\_queue

```

1  priority_queue：優先隊列，資料預設由大到小排序，
   即優先權高的資料會先被取出。
2  宣告：
3      priority_queue<int> pq;
4  把元素 x 加進 priority_queue：
5      pq.push(x);
6  讀取優先權最高的值：
7      x = pq.top();
8      pq.pop(); //讀取後刪除
9  判斷是否為空的priority_queue：
10     pq.empty() //回傳 true
11     pq.size() //回傳 0
12 如需改變priority_queue的優先權定義：
13     priority_queue<T> pq; //預設由大到小
14     priority_queue<T, vector<T>, greater<T>> pq; //改成由小到大
15
16     priority_queue<T, vector<T>, cmp> pq; //cmp

```

### 4.2 queue

```

1  queue：佇列，資料有「先進先出」(first in first out,
   FIFO)的特性。
2  就像排隊買票一樣，先排隊的客戶被服務。
3  宣告：
4      queue<int> q;
5  把元素 x 加進 queue：
6      q.push(x);
7  取值：
8      x = q.front(); //頭
9      x = q.back(); //尾
10  移除已經讀取的值：
11      q.pop();
12  判斷是否為空的queue：
13      q.empty() 回傳 true
14      q.size() 回傳零
15
16 #include <iostream>

```

```

17 #include <queue>
18 using namespace std;
19
20 int main() {
21     int n;
22     while (cin >> n){
23         if (n == 0) break;
24         queue<int> q;
25         for (int i = 0; i < n; i++){
26             q.push(i+1);
27         }
28         cout << "Discarded cards:";
29         for (int i = 0; i < n-1; i++){
30             if (i != 0) cout << ',';
31             cout << ' ' << q.front();
32             q.pop();
33             q.push(q.front());
34             q.pop();
35         }
36         cout << endl << "Remaining card: " <<
37             q.front() << endl;
38     }
39 }

```

### 4.3 deque

```

1  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。
2  dq.push_back() //在 deque 的最尾端新增元素
3  dq.push_front() //在 deque 的開頭新增元素
4  dq.pop_back() //移除 deque 最尾端的元素
5  dq.pop_front() //移除 deque 最開頭的元素
6  dq.back() //取出 deque 最尾端的元素
7  dq.front() //回傳 deque 最開頭的元素
8  dq.insert()
9  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
10 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。
11
12 dq.clear() //清空整個 deque 佇列。
13 dq.size() //檢查 deque 的尺寸
14 dq.empty() //如果 deque 佇列為空返回 1；
   若是存在任何元素，則返回 0
15 dq.begin() //返回一個指向 deque 開頭的迭代器
16 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

```

### 4.4 map

```

1  map：存放 key-value pairs 的映射資料結構，
   會按 key 由小到大排序。
2  元素存取
3  operator[]：存取指定的[i]元素的資料
4
5  迭代器
6  begin()：回傳指向map頭部元素的迭代器
7  end()：回傳指向map末尾的迭代器
8  rbegin()：回傳一個指向map尾部的反向迭代器
9  rend()：回傳一個指向map頭部的反向迭代器
10
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
35 int main(){
36
37     //declaration container and iterator
38     map<string, string> mp;
39     map<string, string>::iterator iter;
40     map<string, string>::reverse_iterator iter_r;
41
42     //insert element
43     mp.insert(pair<string, string>("r000",
44     "student_zero"));
45
46     mp["r123"] = "student_first";
47     mp["r456"] = "student_second";
48
49     //traversal
50     for(iter = mp.begin(); iter != mp.end(); iter++)
51         cout<<iter->first<<" "<<iter->second<<endl;
52     for(iter_r = mp.rbegin(); iter_r != mp.rend();
53         iter_r++)
54         cout<<iter_r->first<<"
55         "<<iter_r->second<<endl;
56
57     //find and erase the element
58     iter = mp.find("r123");
59     mp.erase(iter);
60
61     iter = mp.find("r123");
62
63     if(iter != mp.end())
64         cout<<"Find, the value is
65         "<<iter->second<<endl;
66     else
67         cout<<"Do not Find"<<endl;
68
69     return 0;
70 }
71
72 //map統計數字
73 #include<bits/stdc++.h>
74 using namespace std;
75
76 int main(){
77     ios::sync_with_stdio(0),cin.tie(0);
78     long long n,x;
79     cin>>n;
80     map <int,int> mp;
81     while(n--){
82         cin>>x;
83         ++mp[x];
84     }
85     for(auto i:mp) cout<<i.first<<" "<<i.second<<endl;
86 }

```

## 4.5 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

```

## 4.6 set

```

1 set：集合，去除重複的元素，資料由小到大排序。
2
3 宣告：
4     set <int> st;
5
6 把元素 x 加進 set：
7     st.insert(x);
8
9 檢查元素 x 是否存在 set 中：
10    st.count(x);
11
12 刪除元素 x：
13    st.erase(x); // 可傳入值或iterator
14
15 清空集合中的所有元素：
16    st.clear();
17
18 取值：使用iterator
19    x = *st.begin();
20           // set中的第一個元素(最小的元素)。
21    x = *st.rbegin();
22           // set中的最後一個元素(最大的元素)。
23
24 判斷是否為空的set：
25    st.empty() 回傳true
26    st.size() 回傳零
27
28 常用來搭配的member function：
29    st.count(x);
30    auto it = st.find(x);
31           // binary search, O(log(N))
32    auto it = st.lower_bound(x);
33           // binary search, O(log(N))
34    auto it = st.upper_bound(x);
35           // binary search, O(log(N))

```

## 4.7 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); 會刪除所有值為 val 的元素。
7     st.erase(st.find(val)); 只刪除第一個值為 val
    的元素。

```

## 4.8 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 初始化
7 unordered_set<int> myunordered_set{1, 2, 3, 4, 5};
8
9 陣列初始化
10 int arr[] = {1, 2, 3, 4, 5};
11 unordered_set<int> myunordered_set(arr, arr+5);

```

```

12 |
13 | 插入元素
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 |
22 | int main() {
23 |     unordered_set<int> myunordered_set = {3, 1};
24 |     myunordered_set.insert(2);
25 |     myunordered_set.insert(5);
26 |     myunordered_set.insert(4);
27 |     myunordered_set.insert(5);
28 |     myunordered_set.insert(4);
29 |
30 |     for (const auto &s : myunordered_set) {
31 |         cout << s << " ";
32 |     }
33 |     cout << "\n";
34 |
35 |     return 0;
36 | }
37 |
38 | /*
39 | output
40 | 4 5 2 1 3
41 | */
42 |
43 | unordered_set 刪除指定元素
44 | #include <iostream>
45 | #include <unordered_set>
46 |
47 | int main() {
48 |     unordered_set<int> myunordered_set{2, 4, 6, 8};
49 |
50 |     myunordered_set.erase(2);
51 |     for (const auto &s : myunordered_set) {
52 |         cout << s << " ";
53 |     }
54 |     cout << "\n";
55 |
56 |     return 0;
57 | }
58 | /*
59 | output
60 | 8 6 4
61 | */
62 |
63 | 清空 unordered_set 元素
64 | unordered_set<int> myunordered_set;
65 | myunordered_set.insert(1);
66 | myunordered_set.clear();
67 |
68 | unordered_set 判斷元素是否存在
69 | unordered_set<int> myunordered_set;
70 | myunordered_set.insert(2);
71 | myunordered_set.insert(4);
72 | myunordered_set.insert(6);
73 | cout << myunordered_set.count(4) << "\n"; // 1
74 | cout << myunordered_set.count(8) << "\n"; // 0
75 |
76 | 判斷 unordered_set 容器是否為空
77 | #include <iostream>
78 | #include <unordered_set>
79 |
80 | int main() {
81 |     unordered_set<int> myunordered_set;
82 |     myunordered_set.clear();
83 |
84 |     if (myunordered_set.empty()) {
85 |         cout << "empty\n";
86 |     } else {
87 |         cout << "not empty, size is "<<
            myunordered_set.size() << "\n";

```

```

88 |     }
89 |
90 |     return 0;
91 | }

```

## 4.9 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 1
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 | //寫法1
10 | #include <bits/stdc++.h>
11 | #define maxn 1000100
12 | using namespace std;
13 | int q[maxn], a[maxn];
14 | int n, k;
15 |
16 | void getmin() {
17 |     // 得到這個隊列裡的最小值，直接找到最後的就行了
18 |     int head = 0, tail = 0;
19 |     for (int i = 1; i < k; i++) {
20 |         while (head <= tail && a[q[tail]] >= a[i])
21 |             tail--;
22 |         q[++tail] = i;
23 |     }
24 |     for (int i = k; i <= n; i++) {
25 |         while (head <= tail && a[q[tail]] >= a[i])
26 |             tail--;
27 |         q[++tail] = i;
28 |         while (q[head] <= i - k) head++;
29 |         cout << a[q[head]] << " ";
30 |     }
31 | }
32 | void getmax() { // 和上面同理
33 |     int head = 0, tail = 0;
34 |     for (int i = 1; i < k; i++) {
35 |         while (head <= tail && a[q[tail]] <= a[i]) tail--;
36 |         q[++tail] = i;
37 |     }
38 |     for (int i = k; i <= n; i++) {
39 |         while (head <= tail && a[q[tail]] <= a[i]) tail--;
40 |         q[++tail] = i;
41 |         while (q[head] <= i - k) head++;
42 |         cout << a[q[head]] << " ";
43 |     }
44 | }
45 | int main() {
46 |     cin >> n >> k; //每k個連續的數
47 |     for (int i = 1; i <= n; i++) cin >> a[i];
48 |     getmin();
49 |     cout << '\n';
50 |     getmax();
51 |     cout << '\n';
52 |     return 0;
53 | }
54 |
55 | //寫法2
56 | #include <iostream>
57 | #include <cstring>
58 | #include <deque>
59 | using namespace std;
60 | int a[1000005];
61 |
62 | int main() {
63 |     ios_base::sync_with_stdio(0);
64 |     int n, k;
65 |     while (cin >> n >> k) {
66 |         for (int i = 0; i < n; i++) cin >> a[i];

```

```

67 deque<int> dq;
68 for(int i=0; i<n; i++){
69     while(dq.size() && dq.front()<=i-k)
70         dq.pop_front();
71     while(dq.size() && a[dq.back()]>a[i])
72         dq.pop_back();
73     dq.push_back(i);
74     if(i==k-1) cout<<a[dq.front()];
75     if(i>k-1) cout<<' '<<a[dq.front()];
76 }
77 if(k>n) cout<<a[dq.front()];
78 cout<<'\\n';
79 while(dq.size()) dq.pop_back();
80 for(int i=0; i<n; i++){
81     while(dq.size() && dq.front()<=i-k)
82         dq.pop_front();
83     while(dq.size() && a[dq.back()]<a[i])
84         dq.pop_back();
85     dq.push_back(i);
86     if(i==k-1) cout<<a[dq.front()];
87     if(i>k-1) cout<<' '<<a[dq.front()];
88 }
89 if(k>n) cout<<a[dq.front()];
90 cout<<'\\n';
91 }
92 return 0;
93 }

```

#### example 2

一個含有  $n$  項的數列，求出每一項前的  $m$  個數到它這個區間內的最小值。  
若前面的數不足  $m$  項則從第 1 個數開始，若前面沒有數則輸出 0

```

100 #include<bits/stdc++.h>
101 using namespace std;
102 #define re register int
103 #define INF 0x3f3f3f3f
104 #define ll long long
105 #define maxn 2000009
106 #define maxm
107 inline ll read() {
108     ll x=0,f=1;
109     char ch=getchar();
110     while(ch<'0' || ch>'9'){
111         if(ch=='-') f=-1;
112         ch=getchar();
113     }
114     while(ch>='0' && ch<='9'){
115         x=(x<<1)+(x<<3)+(ll)(ch-'0');
116         ch=getchar();
117     }
118     return x*f;
119 }
120 }
121 int n,m,k,tot,head,tail;
122 int a[maxn],q[maxn];
123 int main() {
124     n=read(), m=read();
125     for(int i=1; i<=n; i++) a[i]=read();
126     head=1, tail=0; //起始位置為1
127     //因為插入是q[++tail]所以要初始化為0
128     for(int i=1; i<=n; i++)
129     //每次隊首的元素就是當前的答案
130     {
131         cout<<a[q[head]]<<endl;
132         while(i-q[head]+1>m && head<=tail) //維護隊首
133             head++;
134         while(a[i]<a[q[tail]] && head<=tail) //維護隊尾
135             tail--;
136         q[++tail]=i;
137     }
138     return 0;
139 }

```

## 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

```

### 5.2 bubble sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin>>n;
7     int a[n], tmp;
8     for(int i=0; i<n; i++) cin>>a[i];
9     for(int i=n-1; i>0; i--) {
10         for(int j=0; j<=i-1; j++) {
11             if( a[j]>a[j+1]) {
12                 tmp=a[j];
13                 a[j]=a[j+1];
14                 a[j+1]=tmp;
15             }
16         }
17     }
18     for(int i=0; i<n; i++) cout<<a[i]<<" ";
19 }

```

## 6 math

### 6.1 質數與因數

```

1 質數
2
3 一般篩法 O(NloglogN)
4 vector<int> p;
5 bitset<MAXN> is_notp;
6 void PrimeTable(int n)
7 {
8     is_notp.reset();
9     is_notp[0] = is_notp[1] = 1;
10    for (int i = 2; i <= n; i++)
11    {
12        if (is_notp[i])
13            continue;
14        p.push_back(i);
15        for (int j = i * i; j <= n; j += i)
16            is_notp[j] = 1;
17    }
18 }
19
20 }
21
22 線性篩法 O(N)
23 vector<int> p;
24 bitset<MAXN> is_notp;
25 void PrimeTable(int n)
26 {

```

```

27 |     is_notp.reset();
28 |     is_notp[0] = is_notp[1] = 1;
29 |     for (int i = 2; i <= n; ++i)
30 |     {
31 |         if (!is_notp[i])
32 |             p.push_back(i);
33 |         for (int j = 0; j < (int)p.size(); ++j)
34 |         {
35 |             if (i * p[j] > n)
36 |                 break;
37 |             is_notp[i * p[j]] = 1;
38 |             if (i % p[j] == 0)
39 |                 break;
40 |         }
41 |     }
42 | }

```

## 因數

最大公因數  $O(\log(\min(a, b)))$

```

47 | int GCD(int a, int b)
48 | {
49 |     if (b == 0) return a;
50 |     return GCD(b, a % b);
51 | }

```

## 質因數分解

```

54 | void primeFactorization(int n)
55 | {
56 |     for (int i = 0; i < (int)p.size(); ++i)
57 |     {
58 |         if (p[i] * p[i] > n)
59 |             break;
60 |         if (n % p[i])
61 |             continue;
62 |         cout << p[i] << ' ';
63 |         while (n % p[i] == 0)
64 |             n /= p[i];
65 |     }
66 |     if (n != 1)
67 |         cout << n << ' ';
68 |     cout << '\n';
69 | }

```

## 歌德巴赫猜想

solution : 把偶數  $N(6 \leq N \leq 10^6)$  寫成兩個質數的和。

```

73 | #include <iostream>
74 | #include <cstdio>
75 | using namespace std;
76 | #define N 2000000
77 | int ox[N], p[N], pr;
78 |
79 | void PrimeTable(){
80 |     ox[0] = ox[1] = 1;
81 |     pr = 0;
82 |     for (int i = 2; i < N; i++){
83 |         if (!ox[i]) p[pr++] = i;
84 |         for (int j = 0; i*p[j]<N&&j < pr; j++)
85 |             ox[i*p[j]] = 1;
86 |     }
87 | }
88 |
89 | int main(){
90 |     PrimeTable();
91 |     int n;
92 |     while (cin>>n,n){
93 |         int x;
94 |         for (x = 1; x += 2)
95 |             if (!ox[x] && !ox[n - x])break;
96 |         printf("%d = %d + %d\n", n, x, n - x);
97 |     }
98 | }

```

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)。

```

102 | 如果  $N$  是奇數且  $N-2$  是質數，則答案為  $2(2 + \text{質數})$ 。
103 | 其他狀況答案為 3(弱歌德巴赫猜想)。
104 | #pragma GCC optimize("O2")
105 | #include <bits/stdc++.h>
106 | using namespace std;
107 | #define FOR(i, L, R) for (int i = L; i < (int)R; ++i)
108 | #define FORD(i, L, R) for (int i = L; i > (int)R; --i)
109 | #define IOS
110 |     cin.tie(nullptr);
111 |     cout.tie(nullptr);
112 |     ios_base::sync_with_stdio(false);
113 |
114 | bool isPrime(int n)
115 | {
116 |     FOR(i, 2, n)
117 |     {
118 |         if (i * i > n)
119 |             return true;
120 |         if (n % i == 0)
121 |             return false;
122 |     }
123 |     return true;
124 | }
125 |
126 | int main()
127 | {
128 |     IOS;
129 |     int n;
130 |     cin >> n;
131 |     if (isPrime(n))
132 |         cout << "1\n";
133 |     else if (n % 2 == 0 || isPrime(n - 2))
134 |         cout << "2\n";
135 |     else
136 |         cout << "3\n";
137 | }

```

## 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 }

```

## 7 algorithm

### 7.1 basic

```

1 min: 取最小值。
2 min(a, b)
3 min(list)
4 max: 取最大值。
5 max(a, b)
6 max(list)
7 min_element: 找尋最小元素
8 min_element(first, last)
9 max_element: 找尋最大元素
10 max_element(first, last)
11 sort: 排序, 預設由小排到大。
12 sort(first, last)
13 sort(first, last, comp): 可自行定義比較運算子 Comp。
14 find: 尋找元素。
15 find(first, last, val)
16 lower_bound: 尋找第一個小於 x
    的元素位置, 如果不存在, 則回傳 last。
17 lower_bound(first, last, val)
18 upper_bound: 尋找第一個大於 x
    的元素位置, 如果不存在, 則回傳 last。
19 upper_bound(first, last, val)
20 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。
21 next_permutation(first, last)
22 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。
23 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  
 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。  
 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 Scarecrow  
 10 //problem  
 11 有一個  $N \times 1$  的稻田，有些稻田現在有種植作物，  
 12 為了避免被動物破壞，需要放置稻草人，  
 13 稻草人可以保護該塊稻田和左右兩塊稻田，  
 14 請問最少需要多少稻草人才能保護所有稻田？

15 //solution  
 16 從左到右掃描稻田，如果第  $i$  塊稻田有作物，  
 17 就把稻草人放到第  $i+1$  塊稻田，  
 18 這樣能保護第  $i, i+1, i+2$  塊稻田，  
 19 接著從第  $i+3$  塊稻田繼續掃描。

20 //code  
 21 #include <bits/stdc++.h>  
 22 using namespace std;  
 23 int main(){  
 24 string s;  
 25 int i, n, t, tc = 1;  
 26 cin >> t;  
 27 while (t--){  
 28 cin >> n >> s;  
 29 int nc = 0;  
 30 for (i = 0; i < n; i++){  
 31 if (s[i] == '.') i += 2, nc++;  
 32 }  
 33 cout << "Case " << tc++ << ": " << nc << endl;  
 34 }  
 35 }  
 36 }  
 37

霍夫曼樹的變形題

38 //problem  
 39 給定  $N$  個數，每次將兩個數  $a, b$  合併成  $a+b$ ，  
 40 只到最後只剩一個數，合併成本為兩數和，  
 41 問最小合併成本為多少。

42 //solution  
 43 每次將最小的兩數合併起來。

44 //code  
 45 #include <bits/stdc++.h>  
 46 using namespace std;  
 47 int main()  
 48 {  
 49 int n, x;  
 50 while (cin >> n, n)  
 51 {  
 52 priority\_queue<int, vector<int>, greater<int>>  
 53 q;  
 54 while (n--)  
 55 {  
 56 cin >> x;  
 57 }  
 58 }  
 59 }

```

59     q.push(x);
60 }
61 long long ans = 0;
62 while (q.size() > 1)
63 {
64     x = q.top();
65     q.pop();
66     x += q.top();
67     q.pop();
68     q.push(x);
69     ans += x;
70 }
71 cout << ans << endl;
72 }
73 }
74

```

75 Commando War  
 76 //problem  
 77 有  $n$  個部下，每個部下要花  $B_i$  分鐘交待任務，  
 78 再花  $J_i$  分鐘執行任務，一次只能對一位部下交代任務，  
 79 但可以多人同時執行任務，問最少要花多少時間完成任務。

80 //solution  
 81 執行時間長的人先交代任務

82 //code  
 83 #include <bits/stdc++.h>  
 84 using namespace std;  
 85 struct Data{  
 86 int b, j;  
 87 bool operator<(const Data &rhs) const {  
 88 return j > rhs.j;  
 89 }  
 90 };  
 91  
 92 int main(){  
 93 int n, ti = 0;  
 94 Data a[1005];  
 95 while (cin >> n, n){  
 96 for (int i = 0; i < n; ++i)  
 97 cin >> a[i].b >> a[i].j;  
 98 sort(a, a + n);  
 99 int ans = 0, sum = 0;  
 100 for (int i = 0; i < n; ++i){  
 101 sum += a[i].b;  
 102 ans = max(ans, sum + a[i].j);  
 103 }  
 104 cout << "Case " << ++ti << ": " << ans << '\n';  
 105 }  
 106 }  
 107 }  
 108 }  
 109

刪數字問題

110 //problem  
 111 給定一個數字  $N (\leq 10^{100})$ ，需要刪除  $K$  個數字，  
 112 請問刪除  $K$  個數字後最小的數字為何？

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

116 //code  
 117 int main()  
 118 {  
 119 string s;  
 120 int k;  
 121 cin >> s >> k;  
 122 for (int i = 0; i < k; ++i)  
 123 {  
 124 if ((int)s.size() == 0)  
 125 break;  
 126 int pos = (int)s.size() - 1;  
 127 for (int j = 0; j < (int)s.size() - 1; ++j)  
 128 {  
 129 if (s[j] > s[j + 1])  
 130 {  
 131 pos = j;  
 132 }  
 133 }  
 134 }  
 135 }



```

135         break;
136     }
137 }
138 s.erase(pos, 1);
139 }
140 while ((int)s.size() > 0 && s[0] == '0')
141     s.erase(0, 1);
142 if ((int)s.size())
143     cout << s << '\n';
144 else
145     cout << 0 << '\n';
146 }
147
148 區間覆蓋長度
149 //problem
150 給定 n 條線段區間為 [Li,Ri]，
151 請問這些線段的覆蓋所覆蓋的長度？
152
153 //solution
154 先將所有區間依照左界由小到大排序，
155 左界相同依照右界由小到大排序，
156 用一個變數 R 紀錄目前最大可以覆蓋到的右界。
157 如果目前區間左界 ≤ R，代表該區間可以和前面的線段合併。
158
159 //code
160 struct Line
161 {
162     int L, R;
163     bool operator<(const Line &rhs) const
164     {
165         if (L != rhs.L)
166             return L < rhs.L;
167         return R < rhs.R;
168     }
169 };
170
171 int main(){
172     int n;
173     Line a[10005];
174     while (cin >> n){
175         for (int i = 0; i < n; i++){
176             cin >> a[i].L >> a[i].R;
177             sort(a, a + n);
178             int ans = 0, L = a[0].L, R = a[0].R;
179             for (int i = 1; i < n; i++){
180                 if (a[i].L < R) R = max(R, a[i].R);
181                 else{
182                     ans += R - L;
183                     L = a[i].L;
184                     R = a[i].R;
185                 }
186             }
187             cout << ans + (R - L) << '\n';
188         }
189     }
190 }
191
192 最小區間覆蓋長度
193 //problem
194 給定 n 條線段區間為 [Li,Ri]，
195 請問最少要選幾個區間才能完全覆蓋 [0,S]？
196
197 //solution
198 先將所有區間依照左界由小到大排序，
199 對於當前區間 [Li,Ri]，要從左界 > Ri 的所有區間中，
200 找到有著最大的右界的區間，連接當前區間。
201
202 //problem
203 長度 n 的直線中有數個加熱器，
204 在 x 的加熱器可以讓 [x-r, x+r] 內的物品加熱，
205 問最少要幾個加熱器可以把 [0, n] 的範圍加熱。
206
207 //solution
208 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。

```

210
211 //code
212 int main(){
213     int n, r;
214     int a[1005];
215     cin >> n >> r;
216     for (int i=1; i<=n; ++i) cin >> a[i];
217     int i = 1, ans = 0;
218     while (i <= n){
219         int R = min(i+r-1, n), L = max(i-r+1, 0);
220         int nextR = -1;
221         for (int j = R; j >= L; --j){
222             if (a[j]){
223                 nextR = j;
224                 break;
225             }
226         }
227         if (nextR == -1){
228             ans = -1;
229             break;
230         }
231         ++ans;
232         i = nextR + r;
233     }
234     cout << ans << '\n';
235 }
236
237
238 最多不重疊區間
239 //problem
240 給你 n 條線段區間為 [Li,Ri]，
241 請問最多可以選擇幾條不重疊的線段(頭尾可相連)？
242
243 //solution
244 依照右界由小到大排序，
245 每次取到一個不重疊的線段，答案 +1。
246
247 //code
248 struct Line
249 {
250     int L, R;
251     bool operator<(const Line &rhs) const {
252         return R < rhs.R;
253     }
254 };
255
256 int main()
257 {
258     int t;
259     cin >> t;
260     Line a[30];
261     while (t--){
262         {
263             int n = 0;
264             while (cin >> a[n].L >> a[n].R, a[n].L || a[n].R)
265                 ++n;
266             sort(a, a + n);
267             int ans = 1, R = a[0].R;
268             for (int i = 1; i < n; i++){
269                 if (a[i].L >= R)
270                 {
271                     ++ans;
272                     R = a[i].R;
273                 }
274             }
275             cout << ans << '\n';
276         }
277     }
278 }
279
280 區間選點問題
281 //problem
282 給你 n 條線段區間為 [Li,Ri]，
283 請問至少要取幾個點才能讓每個區間至少包含一個點？

```

```

286
287 //solution
288 將區間依照右界由小到大排序，R=第一個區間的右界，
289 遍歷所有區段，如果當前區間左界>R，
290 代表必須多選一個點 (ans+=1)，並將 R=當前區間右界。
291
292 //problem
293 給定 N 個座標，要在 x 軸找到最小的點，
294 讓每個座標至少和一個點距離 ≤ D。
295
296 //solution
297 以每個點 (xi,yi) 為圓心半徑為 D 的圓 C，
298 求出 C 和 x 軸的交點 Li,Ri，題目轉變成區間選點問題。
299
300 //code
301 struct Line
302 {
303     int L, R;
304     bool operator<(const Line &rhs) const {
305         return R < rhs.R;
306     }
307 };
308
309 int main()
310 {
311     int t;
312     cin >> t;
313     Line a[30];
314     while (t--)
315     {
316         int n = 0;
317         while (cin>>a[n].L>>a[n].R, a[n].L||a[n].R)
318             ++n;
319         sort(a, a + n);
320         int ans = 1, R = a[0].R;
321         for (int i = 1; i < n; i++)
322         {
323             if (a[i].L >= R)
324             {
325                 ++ans;
326                 R = a[i].R;
327             }
328         }
329         cout << ans << '\n';
330     }
331 }
332
333 最小化最大延遲問題
334 //problem
335 給定 N 項工作，每項工作的需要處理時長為 Ti，
336 期限是 Di，第 i 項工作延遲的時間為 Li=max(0,Fi-Di)，
337 原本Fi 為第 i 項工作的完成時間，
338 求一種工作排序使 maxLi 最小。
339
340 //solution
341 按照到期時間從早到晚處理。
342
343 //code
344 struct Work
345 {
346     int t, d;
347     bool operator<(const Work &rhs) const {
348         return d < rhs.d;
349     }
350 };
351
352 int main()
353 {
354     int n;
355     Work a[10000];
356     cin >> n;
357     for (int i = 0; i < n; ++i)
358         cin >> a[i].t >> a[i].d;
359     sort(a, a + n);
360     int maxL = 0, sumT = 0;

```

```

362     for (int i = 0; i < n; ++i)
363     {
364         sumT += a[i].t;
365         maxL = max(maxL, sumT - a[i].d);
366     }
367     cout << maxL << '\n';
368 }
369
370 最少延遲數量問題
371 //problem
372 給定 N 個工作，每個工作的需要處理時長為 Ti，
373 期限是 Di，求一種工作排序使得逾期工作數量最小。
374
375 //solution
376 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，
377 依序放入工作列表中，如果發現有工作預期，
378 就從目前選擇的工作中，移除耗時最長的工作。
379
380 上述方法為 Moore-Hodgson s Algorithm。
381
382 //problem
383 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
384
385 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
386
387 工作處理時長 → 烏龜重量
388 工作期限 → 烏龜可承受重量
389 多少工作不延期 → 可以疊幾隻烏龜
390
391 //code
392 struct Work{
393     int t, d;
394     bool operator<(const Work &rhs) const {
395         return d < rhs.d;
396     }
397 };
398
399 int main()
400 {
401     int n = 0;
402     Work a[10000];
403     priority_queue<int> pq;
404     while(cin >> a[n].t >> a[n].d)
405         ++n;
406     sort(a, a + n);
407     int sumT = 0, ans = n;
408     for (int i = 0; i < n; ++i)
409     {
410         pq.push(a[i].t);
411         sumT += a[i].t;
412         if(a[i].d<sumT)
413         {
414             int x = pq.top();
415             pq.pop();
416             sumT -= x;
417             --ans;
418         }
419     }
420     cout << ans << '\n';
421 }
422
423 任務調度問題
424 //problem
425 給定 N 項工作，每項工作的需要處理時長為 Ti，
426 期限是 Di，如果第 i 項工作延遲需要受到 pi 單位懲罰，
427 請問最少會受到多少單位懲罰。
428
429 //solution
430 依照懲罰由大到小排序，
431 每項工作依序嘗試可不可以放在 Di-Ti+1,Di-Ti,...,1,0，
432 如果有空閒就放進去，否則延後執行。
433
434 //problem
435 給定 N 項工作，每項工作的需要處理時長為 Ti，

```

437 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，

438 請問最多會獲得多少單位獎勵。

439  
440 *//solution*  
441 和上題相似，這題變成依照獎勵由大到小排序。

```
442 //code
443 struct Work
444 {
445     int d, p;
446     bool operator<(const Work &rhs) const {
447         return p > rhs.p;
448     }
449 };
450
451 int main()
452 {
453     int n;
454     Work a[100005];
455     bitset<100005> ok;
456     while (cin >> n)
457     {
458         ok.reset();
459         for (int i = 0; i < n; ++i)
460             cin >> a[i].d >> a[i].p;
461         sort(a, a + n);
462         int ans = 0;
463         for (int i = 0; i < n; ++i)
464         {
465             int j = a[i].d;
466             while (j-->0)
467                 if (!ok[j])
468                 {
469                     ans += a[i].p;
470                     ok[j] = true;
471                     break;
472                 }
473             }
474         }
475         cout << ans << '\n';
476     }
477 }
```

478 多機調度問題

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

484 *//solution*  
485 將工作由大到小排序，每項工作交給最快空閒的機器。

```
486 //code
487 int main()
488 {
489     int n, m;
490     int a[10000];
491     cin >> n >> m;
492     for (int i = 0; i < n; ++i)
493         cin >> a[i];
494     sort(a, a + n, greater<int>());
495     int ans = 0;
496     priority_queue<int, vector<int>, greater<int>> > pq;
497     for (int i = 0; i < m && i < n; ++i)
498     {
499         ans = max(ans, a[i]);
500         pq.push(a[i]);
501     }
502     for (int i = m; i < n; ++i)
503     {
504         int x = pq.top();
505         pq.pop();
506         x += a[i];
507         ans = max(ans, x);
508         pq.push(x);
509     }
510 }
```

```
512     cout << ans << '\n';
513 }
```

## 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.
18 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
19 5. 如果有數字在 B 裡面而不在 A
20 裡面，直接忽略這個數字不做轉換即可
```

## 9 graph

### 9.1 graph

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 class Node {
5 public:
6     int val;
7     vector<Node*> children;
8
9     Node() {}
10
11     Node(int _val) {
12         val = _val;
13     }
14
15     Node(int _val, vector<Node*> _children) {
16         val = _val;
17         children = _children;
18     }
19 };
20
21 struct ListNode {
22     int val;
23     ListNode *next;
24     ListNode() : val(0), next(nullptr) {}
25     ListNode(int x) : val(x), next(nullptr) {}
26     ListNode(int x, ListNode *next) : val(x),
27         next(next) {}
28 };
29
30 struct TreeNode {
31     int val;
32     TreeNode *left;
33     TreeNode *right;
34     TreeNode() : val(0), left(nullptr),
35         right(nullptr) {}
36     TreeNode(int x) : val(x), left(nullptr),
37         right(nullptr) {}
38 }
```

```

35     TreeNode(int x, TreeNode *left, TreeNode *right)
36         : val(x), left(left), right(right) {}
37 };
38 class ListProblem {
39     vector<int> nums={};
40 public:
41     void solve() {
42         return;
43     }
44
45     ListNode* buildList(int idx) {
46         if(idx == nums.size()) return NULL;
47         ListNode *current=new
48             ListNode(nums[idx++],current->next);
49         return current;
50     }
51
52     void deleteList(ListNode* root) {
53         if(root == NULL) return;
54         deleteList(root->next);
55         delete root;
56         return;
57     }
58
59     class TreeProblem {
60         int null = INT_MIN;
61         vector<int> nums = {}, result;
62     public:
63         void solve() {
64
65             return;
66         }
67
68         TreeNode* buildBinaryTreeUsingDFS(int left, int
69             right) {
70             if((left > right) || (nums[(left+right)/2] ==
71                 null)) return NULL;
72             int mid = (left+right)/2;
73             TreeNode* current = new TreeNode(
74                 nums[mid],
75                 buildBinaryTreeUsingDFS(left,mid-1),
76                 buildBinaryTreeUsingDFS(mid+1,right));
77             return current;
78         }
79
80         TreeNode* buildBinaryTreeUsingBFS() {
81             int idx = 0;
82             TreeNode* root = new TreeNode(nums[idx++]);
83             queue<TreeNode*> q;
84             q.push(root);
85             while(idx < nums.size()) {
86                 if(nums[idx] != null) {
87                     TreeNode* left = new
88                         TreeNode(nums[idx]);
89                     q.front()->left = left;
90                     q.push(left);
91                 }
92                 idx++;
93                 if((idx < nums.size()) && (nums[idx] !=
94                     null)) {
95                     TreeNode* right = new
96                         TreeNode(nums[idx]);
97                     q.front()->right = right;
98                     q.push(right);
99                 }
100                 idx++;
101                 q.pop();
102             }
103             return root;
104         }
105
106         Node* buildNaryTree() {
107             int idx = 2;
108             Node *root = new Node(nums.front());
109             queue<Node*> q;

```

```

105     q.push(root);
106     while(idx < nums.size()) {
107         while((idx < nums.size()) && (nums[idx]
108             != null)) {
109             Node *current = new Node(nums[idx++]);
110             q.front()->children.push_back(current);
111             q.push(current);
112         }
113         idx++;
114         q.pop();
115     }
116     return root;
117 }
118
119 void deleteBinaryTree(TreeNode* root) {
120     if(root->left != NULL)
121         deleteBinaryTree(root->left);
122     if(root->right != NULL)
123         deleteBinaryTree(root->right);
124     delete root;
125     return;
126 }
127
128 void deleteNaryTree(Node* root) {
129     if(root == NULL) return;
130     for(int i=0; i<root->children.size(); i++) {
131         deleteNaryTree(root->children[i]);
132         delete root->children[i];
133     }
134     delete root;
135     return;
136 }
137
138 void inorderTraversal(TreeNode* root) {
139     if(root == NULL) return;
140     inorderTraversal(root->left);
141     cout<<root->val<< ' ';
142     inorderTraversal(root->right);
143     return;
144 }
145
146 int main() {
147     return 0;
148 }

```

## 10 Section2

### 10.1 thm

- 中文測試

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