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

7 algorithm 5
7.1 basic . . . 5
7.2 binary search . . . 5
7.3 prefix sum . . . 5
7.4 差分 . . . 5
7.5 greedy . . . 6
7.6 floyd warshall . . . 7
7.7 dinic . . . 7
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 10
9.1 thm . . . 10

1 ubuntu

1.1 run

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

1.2 cp.sh

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

2 Basic

2.1 ascii

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

2.2 limits

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

3 字串

3.1 最長迴文子字串

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

string s;
int n;

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

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

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

3.2 stringstream

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

4 STL

4.1 priority_queue

```
priority_queue： 優先隊列，資料預設由大到小排序。

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

4.2 deque

```
deque 是 C++ 標準模板函式庫
     (Standard Template Library, STL)
    中的雙向佇列容器 (Double-ended Queue) ，
    跟 vector 相似，不過在 vector
        中若是要添加新元素至開端，
    其時間複雜度為 O(N)，但在 deque 中則是 O(1)。
    同樣也能在我們需要儲存更多元素的時候自動擴展空間，
    讓我們不必煩惱佇列長度的問題。

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

4.3 map

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

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

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

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

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

查找
count()：回傳指定元素出現的次數
find()：查找一個元素

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

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

    //traversal
```

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

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

4.4 unordered_map

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

4.5 set

```
set： 集合，去除重複的元素，資料由小到大排序。

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

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

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

4.6 multiset

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

4.7 unordered_set

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

unordered_set 判斷元素是否存在
```

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

4.8 單調隊列

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

example

給出一個長度為 n 的數組，
輸出每 k 個連續的數中的最大值和最小值。

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

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

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

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

5 sort

5.1 大數排序

```
#python大數排序

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

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

6 math

6.1 質數與因數

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

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

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

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

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

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

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



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

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

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

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

6.2 快速冪

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

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

```
    }
    return ans;
}

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

6.3 歐拉函數

```
//計算閉區間 [1,n] 中的正整數與 n 互質的個數

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

7 algorithm

7.1 basic

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

7.2 binary search

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

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

7.3 prefix sum

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

#include<bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin>>n;
    int a[n],b[n];
    for(int i=0;i<n;i++) cin>>a[i];
    b[0]=a[0];
    for(int i=1;i<n;i++) b[i]=b[i-1]+a[i];
    for(int i=0;i<n;i++) cout<<b[i]<<' ';
    cout<<'\n';
    int l,r;
    cin>>l>>r;
    cout<<b[r]-b[l-1]; //區間和
}
```

7.4 差分

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

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

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

7.5 greedy

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


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

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

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


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

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

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

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

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


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

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

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

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


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

//solution
按照到期時間從早到晚處理。

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

int main(){
    int n;
    Work a[10000];
    cin>>n;
    for(int i=0;i<n;++i)
```

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


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

//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。
工作處裡時長 → 烏龜重量
工作期限 → 烏龜可承受重量
多少工作不延期 → 可以疊幾隻烏龜

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

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

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

//solution
依照懲罰由大到小排序，
每項工作依序嘗試可不可以放在 Di-Ti+1,Di-Ti,...,1,0，
如果有空閒就放進去，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 Ti，
期限是 Di，如果第 i 項工作在期限內完成會獲得 ai
    單位獎勵，
請問最多會獲得多少單位獎勵。

//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';
    }
}
```

7.6 floyd warshall

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

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

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

7.7 dinic

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

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

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

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

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

7.8 SegmentTree

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

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

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

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

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

```
    if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
    if (qr > mid) update(ql, qr, mid+1, r, i*2+1, c);
    st[i] = pull(i * 2, i * 2 + 1);
}
//如果是直接改值而不是加值，query與update中的tag與st的
//改值從+=改成=
```

7.9 Nim Game

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

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

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

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

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

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

7.10 Trie

```
#include <bits/stdc++.h>
#define word_maxn 4000*100+5
#define str_maxn 300000+5
#define sigma_num 26
#define MOD 20071027
using namespace std;

int dp[str_maxn];
char S[str_maxn];
char wd[100+5];

struct Trie{
    int ch[word_maxn][sigma_num];
    int val[word_maxn];
    int seq;
    void init(){
        seq=1;
        memset(ch,0,sizeof(ch));
    }
    void insertion(char *s){
        int row=0,n=strlen(s);
        for(int i=0;i<n;i++){
            int letter_no=s[i]-'a';
            if(ch[row][letter_no]==0){
                ch[row][letter_no]=seq;
                memset(ch[seq],0,sizeof(ch[seq]));
                val[seq++]=0;
            }
            row=ch[row][letter_no];
        }
        val[row]=n;
    }
    void find_prefix(char *s,int len,vector<int>&vc){
        int row=0;
        for(int i=0;i<len;i++){
            int letter_no=s[i]-'a';
            if(ch[row][letter_no]==0) return;
            row=ch[row][letter_no];
            if(val[row]) vc.push_back(val[row]);
        }
    }
}tr;

int main(){
    int Case=1;
    while(cin>>S){
        int n;
        cin>>n;
        tr.init();
        for(int i=0;i<n;i++){
            cin>>wd;
            tr.insertion(wd);
        }
        memset(dp,0,sizeof(dp));
        int N=strlen(S);
        dp[N]=1;
        for(int i=N-1;i>=0;i--){
            vector<int> vc;
            tr.find_prefix(S+i,N-i,vc);
            for(int j=0;j<vc.size();j++)
                dp[i]=(dp[i]+dp[i+vc[j]])%MOD;
        }
        cout<<"Case "<<Case++<<": "<<dp[0]<<endl;
    }
    return 0;
}

/*
 input
abcd
4
a b cd ab
```

```
 output
Case 1: 2
*/
```

7.11 SPFA

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

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

7.12 dijkstra

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

struct edge{
    int v,w;
};

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

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

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

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

8 動態規劃

8.1 LCS 和 LIS

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

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

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

9 Section2

9.1 thm

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