Contents

1 Math

1.1 快速冪

```
/*快速冪*/
ll mypow(ll x, ll y, ll p) {
    long long ans = 1;
    while (y) {
        if (y & 1)    ans = ans * x % p;    //prime
        x = x * x % p;    //每次把自己平方
        y >>= 1;    //每次右移一格
    }
    return ans;
}
```

1.2 快速乘

```
/*快速乘(a * b) mod m 大數乘法取餘數*/
ll mul(ll x, ll y, ll mod) {
    ll ret = x * y - (ll)((long double)x / mod * y) *
        mod;
    // LL ret=x*y-(LL)((long double)x*y/mod+0.5)*mod;
    return ret < 0 ? ret + mod : ret;
}
```

1.3 快速乘法 karatsuba

```
/*karatsuba 快速乘法*/

// Get size of the numbers
int getSize(ll num)
{
    int count = 0;
    while (num > 0)
    {
        count++;
        num /= 10;
    }
    return count;
}

ll karatsuba(ll X, ll Y){
    // Base Case
    if (X < 10 && Y < 10)
        return X * Y;

    // determine the size of X and Y
    int size = fmax(getSize(X), getSize(Y));

    // Split X and Y
    int n = (int)ceil(size / 2.0);
    ll p = (ll)pow(10, n);
    ll a = (ll)floor(X / (double)p);
    ll b = X % p;
    ll c = (ll)floor(Y / (double)p);
    ll d = Y % p;

    // Recur until base case
    ll ac = karatsuba(a, c);
    ll bd = karatsuba(b, d);
    ll e = karatsuba(a + b, c + d) - ac - bd;

    // return the equation
    return (ll)(pow(10 * 1L, 2 * n) * ac + pow(10 * 1L,
         n) * e + bd);
}
```

1.4 GCD

```
/*GCD*/
ll gcd(ll a, ll b){
    return b == 0 ? a : gcd(b, a % b);
}
```

1.5 ax+by=gcd(a,b)

```
/*ax+by=gcd(a,b) 一組解*/
ll a, b, x, y;
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    }
    return x = 1, y = 0, a;
}
```

1.6 Chinese Remainder Theorem

```
/*Chinese remainder theorem*/
ll CRT(int k, ll* a, ll* r) {
  ll n = 1, ans = 0;
  for (int i = 1; i <= k; i++) n = n * r[i];
  for (int i = 1; i <= k; i++) {
    ll m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % mod) % mod;
  }
  return (ans % mod + mod) % mod;
}
```

1.7 模反元素 inverse

```
/*Chinese remainder theorem*/
ll CRT(int k, ll* a, ll* r) {
  ll n = 1, ans = 0;
  for (int i = 1; i <= k; i++) n = n * r[i];
  for (int i = 1; i <= k; i++) {
    ll m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % mod) % mod;
  }
  return (ans % mod + mod) % mod;
}
```

1.8 Sieve Prime

```
/*Sieve_Prime*/
const int N = 20000000;//質數表大小
bool sieve[N];
vector<int> prime;
void linear_sieve(){
    for (int i = 2; i < N; i++)
    {
        if (!sieve[i]) prime.push_back(i);
        for (int p : prime)
        {
            if (i * p >= N) break;
            sieve[i * p] = true;
            if (i % p == 0) break;
        }
    }
}
```

1.9 Miller Rabin

```
/*Miller_Rabin 質數判定*/
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
ll magic[N] = {};
bool witness(ll a, ll n, ll u, int t) {
    if (!a) return 0;
    ll x = mypow(a, u, n);  //快速冪
    for (int i = 0; i < t; i++) {
        ll nx = mul(x, x, n);   //快速乘
        if (nx == 1 && x != 1 && x != n - 1) return 1;
        x = nx;
    }
    return x != 1;
}
bool miller_rabin(ll n) {
    int s = (magic number size);
        // iterate s times of witness on n
        if (n < 2) return 0;
    if (!(n & 1)) return n == 2;
    ll u = n - 1; int t = 0;
    // n-1 = u*2^t
    while (!(u & 1)) u >>= 1, t++;
    while (s--) {
        ll a = magic[s] % n;
        if (witness(a, n, u, t)) return 0;
    }
    return 1;
}
```

1.10 Prime factorization

```
/*質因數分解*/
list<int> breakdown(int N) {
  list<int> result;
  for (int i = 2; i * i <= N; i++) {
    if (N % i == 0) {  // 如果 i 能够整除 N，说明 i 为 N 的一个质因子。
      while (N % i == 0) N /= i;
      result.push_back(i);
    }
  }
  if (N != 1) {  // 说明再经过操作之后 N 留下了一个素数
    result.push_back(N);
  }
  return result;
}
```

1.11 Fibonacci

```
/*Fibonacci*/
int Fib[100005];
int F(int n) {
    Fib[0] = 0; Fib[1] = 1;

    for (int i = 2; i <= n; i++)
        Fib[i] = Fib[i - 1] + Fib[i - 2];

    return Fib[n];
}
```

1.12 josephus

```
/*約瑟夫問題：n個人圍成一桌，數到m的人出列*/
int josephus(int n, int m) {  //n人每m次
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        ans = (ans + m) % i;
    return ans;
}
```

1.13 MOD

```
/*MOD*/
/// _fd(a,b)  floor(a/b).
/// _rd(a,m)  a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r }.
int _fd(int a, int b) { return a < 0 ? (-~a / b - 1) : a / b; }
int _rd(int a, int m) { return a - _fd(a, m) * m; }
int _pv(int a, int m, int r) {
    r = (r % m + m) % m;
    return _fd(a - r, m) * m + r;
}
```

```
int _nt(int a, int m, int r) {
    m = abs(m);
    r = (r % m + m) % m;
    return _fd(a - r - 1, m) * m + r + m;
}
int _ct(int a, int b, int m, int r) {
    m = abs(m);
    a = _nt(a, m, r);
    b = _pv(b, m, r);
    return (a > b) ? 0 : ((b - a + m) / m);
}
```

1.14 Epsilon

```
/*精準度(Epsilon)*/
void Equal(float a, float b)    //判斷相等
{
    float eps = 1e-8;
    if ((fabs(a - b)) < eps)
        printf("Yes\n");
    else printf("No\n");
}
void NEqual(float a, float b)   //判斷不相等
{
    float eps = 1e-8;
    if ((fabs(a - b)) > eps)
        printf("Yes\n");
    else printf("No\n");
}
void Less(float a, float b) //判斷小於
{
    float eps = 1e-8;
    if ((a - b) < -eps)
        printf("Yes\n");
    else printf("No\n");
}
void Greater(float a, float b)  //判斷大於
{
    float eps = 1e-8;
    if ((a - b) > eps)
        printf("Yes\n");
    else printf("No\n");
}
```

1.15 取整函數 floor-ceil

```
/*floor向下取整，ceil向上取整*/
int floor(int a,int b){ return a/b - (a%b and a<0^b<0);
     }
int ceil (int a,int b){ return a/b + (a%b and a<0^b>0);
     }
```

1.16 Big number

```
/*大數(Big Number)*/
void add(int a[100], int b[100], int c[100])    //加法
{
    int i = 0, carry = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] + b[i] + carry;
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void sub(int a[100], int b[100], int c[100])    //減法
{
    int i = 0, borrow = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] - b[i] - borrow;
        if (c[i] < 0) {
            borrow = 1;
            c[i] += 10;
        }
        else
            borrow = 0;
    }
}
void mul(int a[100], int b[100], int c[100])    //乘法
{
    int i = 0, j = 0,carry = 0;
    for (i = 0; i < 100; ++i) {
        if (a[i] == 0) continue;
        for (j = 0; j < MAX; ++j)
            c[i + j] += a[i] * b[i];
    }

    for (i = 0; i < MAX; ++i) {
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void div(int a[100], int b[100], int c[100])    //除法
{
    int t[100];

    for (i = 100 - 1; i >= 0; i--) {
        for (int k = 9; k > 0; k--) // 嘗試商數
        {
            mul(b + i, k, t);
            if (largerthan(a + i, t))
            {
                sub(a + i, t, c + i);
                break;
            }
        }
    }
}
```

1.17 GaussElimination

```
/*GaussElimination*/
// by bcw_codebook

const int MAXN = 300;
const double EPS = 1e-8;

int n;
double A[MAXN][MAXN];

void Gauss() {
  for(int i = 0; i < n; i++) {
    bool ok = 0;
    for(int j = i; j < n; j++) {
      if(fabs(A[j][i]) > EPS) {
        swap(A[j], A[i]);
        ok = 1;
        break;
      }
    }
    if(!ok) continue;

    double fs = A[i][i];
    for(int j = i+1; j < n; j++) {
      double r = A[j][i] / fs;
      for(int k = i; k < n; k++) {
        A[j][k] -= A[i][k] * r;
      }
    }
  }
}
```

2 Data structure

2.1 BIT 樹狀數組 (動態前綴和)

```
/*BIT 樹狀數組(動態前綴和)*/
//BIT and Array start at 1
#define MAXN 100005 //最大區間<MAXN
vector <int> arr(MAXN); //原始陣列
vector <int> bit(MAXN); //BIT數組

//前綴和查詢
ll query(int i) { //index
    ll ret = 0;
    while(i > 0) ret += bit[i], i -= i & -i; // 1-base i-lowbit(i)
    return ret;
}

//單點增值
void modify(int i, int val) { //index,value
    while(i <= MAXN) bit[i] += val, i += i & -i; // i+lowbit(i)
}
```

2.2 Segment tree 線段樹 (區間問題)

```
/*Segment tree 線段樹(區間問題)*/
//segment tree and Array start at 1
// [l,r] 最大區間設為[1,n]
// [ql,qr] 目標區間
// pos,val 修改位置,修改值
#define MAXN 100005*4  //tree大小為4n
#define cl(x) (x*2)    //左子節點index
#define cr(x) (x*2+1)  //右子節點index
#define NO_TAG 0 //懶惰記號
vector <int> tag(MAXN);
vector <int> arr(MAXN);
vector <int> tree(MAXN);

void build(int i,int l,int r){ //i為當前節點index，l,r為當前遞迴區間
    if(l == r){ // 遞迴到區間大小為1
        tree[i] = arr[l];
        return;
    }
    int mid=(l+r)/2; //往兩邊遞迴
    build(cl(i),l,mid);
    build(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]); //<-可修改條件
    //將節點的值設成左右子節點的最大值
}

// i 為當前節點index, l, r當前區間左右界，ql, qr詢問左右界
int query(int i,int l,int r,int ql,int qr){
    if(ql <= l && r <= qr){ //若當前區間在詢問區間內，直接回傳區間最大值
        return tree[i];
    }
    int mid=(l+r)/2, ret=0; //<-可修改條件
    if(ql<=mid) // 如果左子區間在詢問區間內
        ret = max(ret, query(cl(i),l,mid,ql,qr));   //<-可修改條件
    if(qr> mid) // 如果右子區間在詢問區間內
        ret = max(ret, query(cr(i),mid+1,r,ql,qr));  //<-可修改條件
    return ret;
}
```

```
/*單點修改*/
void update(int i,int l,int r,int pos,int val){
    if(l == r){ // 修改 a[pos] 的值為 val
        tree[i] = val;
        return;
    }
    int mid=(l+r)/2;
    if(pos <= mid) // 如果修改位置在左子節點，往左遞迴
        update(cl(i),l,mid,pos,val);
    else // 否則往右遞迴
        update(cr(i),mid+1,r,pos,val);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);    //<-可修改條件
}

/*區間修改*/
//將區間 [l, r] 的值都加 v
void push(int i,int l,int r){
    if(tag[i] != NO_TAG){ // 判斷是否有打標記,NO_TAG=0
        tree[i] += tag[i];  // 有的話就更新當前節點的值
        if(l != r){  // 如果有左右子節點把標記往下打
            tag[cl(i)] += tag[i];
            tag[cr(i)] += tag[i];
        }
        tag[i] = NO_TAG; // 更新後把標記消掉
    }
}

void pull(int i,int l,int r){
    int mid = (l+r)/2;
    push(cl(i),l,mid); push(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);
}

void update(int i,int l,int r,int ql,int qr,int v){
    push(i,l,r);
    if(ql<=l && r<=qr){
        tag[i] += v; //將區間 [l, r] 的值都加 v
        return;
    }
    int mid=(l+r)/2;
    if(ql<=mid)  update(cl(i),l,mid,ql,qr,v);
    if(qr> mid)  update(cr(i),mid+1,r,ql,qr,v);
    pull(i,l,r);
}
```

2.3 Heap

```
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a,b;

int main() {
  a.clear();
  b.clear();
  a.push(1);
  a.push(3);
  b.push(2);
  b.push(4);
  assert(a.top() == 3);
  assert(b.top() == 4);
  // merge two heap
  a.join(b);
  assert(a.top() == 4);
  assert(b.empty());

  return 0;
}
```

3 Algorithm

3.1 Binary Search

```
/*Binary Search*/
void binary_search(ll n,ll target){
  ll L=0, R=n;
    while(L<R){
        ll mid=(L+R)>>1;
        if(check(mid))  R=mid;
        else            L=mid+1;
    }
  cout << L << "\n";
}
```

3.2 DFS

```
/*DFS*/

/*n皇后*/
//k為第幾行，a[k]為第幾列，n個皇后
int a[100], n, count;
void DFS(int k) {
    if (k > n) {//當k=n+1時找到解
        count++;
        printf("第%d個解\n", count);
        for (int i = 1; i <= n; i++) {//譜面輸出
            for (int j = 1; j < a[i]; j++)printf("0");
            printf("1");
            for (int j = a[i] + 1; j <= n; j++)printf("
                0");
            printf("\n");
        }
    }
    else {
        for (int i = 1; i <= n; i++) {//找不到合適的列
            （位置）,回到上一行
            a[k] = i;  //存入皇后
            if (check(a, k))DFS(k + 1);//當前皇后的位置
                符合要求，則求下一個皇后(下一行)
        }
    }
}

/*騎士旅行問題*/
#define X 5 //棋盤
#define Y 5
//騎士共有8個方向
int dir
    [8][2]={{1,2},{2,1},{1,-2},{-2,1},{-1,2},{2,-1},{-1,-

int board[X][Y]={0},tot=0,_x,_y;

void print() {
  cout<<++tot<<"\n";
  for (int i=0;i<X;i++) {
    for (int j=0;j<Y;j++)
      cout<<setw(3)<<board[i][j];
    cout<<"\n";
  }
  return;
}

void dfs(int x,int y,int t) {
  if (t>X*Y) {
    print();
    return;
  }
  for (int i=0;i<8;i++) {
    int xx=x+dir[i][0];
    int yy=y+dir[i][1];
    if ((xx>=X)||(xx<0)||(yy>=Y)||(yy<0)||(board[xx][yy
        ])) continue;
    board[xx][yy]=t;
    dfs(xx,yy,t+1);
    board[xx][yy]=0; //回溯
  }
}
```

```
int main(){
  cin>>_x>>_y; //起始點
  board[_x][_y]=1;
  dfs(_x,_y,2);
}
```

3.3 Brute Force 暴力搜尋

```
/*Brute Force*/

#define MAXN 1<<18+5 //雙倍空間
/*折半枚舉 與 二進制枚舉*/
int main() {
    int n, m, i, temp;
    ll mod, mod_max = 0;
    vector<ll> arr, ans(MAXN,0), ans2(MAXN,0);
    cin >> n >> m;
    for(i=0;i<n;i++){
        cin >> temp;
        arr.push_back(temp%m);
    }

    //折半枚舉
    for(int i=0;i<(1<<(n/2));i++){ //2^(n/2)
        for(int j=0;j<n/2;j++){
            if(i>>j&1) //二進制枚舉(選或不選)
                ans[i] = (ans[i] + arr[j]) % m; //前半
                    枚舉
        }
    }
    for(int i=0;i<(1<<(n-n/2));i++){ //2^(n-n/2)
        for(int j=0;j<(n-n/2);j++){
            if(i>>j&1) ans2[i] = (ans2[i] + arr[n/2+j])
                 % m; //後半枚舉
        }
    }

    //二分維護
    temp = 1<<(n-n/2);
    sort(ans2.begin(), ans2.begin() + temp);
    for(auto i:ans){
        mod_max = max(mod_max, i + *(upper_bound(ans2.
            begin(), ans2.begin() + temp, m-1-i)-1));
        //mod最大為m-1，配對另一半最優解
    }
    cout << mod_max <<"\n";

    return 0;
}
```

3.4 Divide and Conquer 分治法

```
/*Divide and Conquer*/

// 最近點對
double dist(pair<double, double> a, pair<double, double
    > b) {
  return sqrt(pow((a.first - b.first), 2) + pow((a.
      second - b.second), 2));
}
double closest(int l, int r) {
  if (l >= r)
    return 10000;
  int mid = (l + r) / 2;
  double radl = closest(l, mid);
  double radr = closest(mid + 1, r);
  double range = min(radl, radr), middle = (point[mid].
      first + point[mid + 1].first) / 2, minimum;
  minimum = range;
  for (int i = mid + 1; i <= r && point[i].first <
      middle + range; i++) {
```

```
    for (int j = mid; j >= l && point[j].first > middle
         - range; j--) {
      minimum = min(minimum, dist(point[i], point[j]));
    }
  }
  return minimum;
}

/*Quick Sort*/
void quicksort(int array[], int left, int right)
{
    if (left >= right) return;

    int i = left, j = right;
    int k = (left + right) / 2; // 可以隨便選
    int pivot = array[k];
    while (true)
    {
        // 小於等於改成小於，就不必偵測陣列邊界。
        while (array[i] < pivot) i++;
        while (array[j] > pivot) j--;
        if (i >= j) return;
        // 代價：等於pivot的數字，一直做交換。
        swap(array[i], array[j]);
        i++; j--;
    }

    // 代價：array[i]不一定就是pivot。
    quicksort(array, left, j);
    quicksort(array, i, right);
}

/*Merge Sort*/
void Merge(vector<int> &Array, int front, int mid, int
    end) {
    vector<int> LeftSubArray(Array.begin() + front,
        Array.begin() + mid + 1);
    vector<int> RightSubArray(Array.begin() + mid + 1,
        Array.begin() + end + 1);
    int idxLeft = 0, idxRight = 0;
    LeftSubArray.insert(LeftSubArray.end(),
        numeric_limits<int>::max());
    RightSubArray.insert(RightSubArray.end(),
        numeric_limits<int>::max());
    // Pick min of LeftSubArray[idxLeft] and
        RightSubArray[idxRight], and put into Array[i]
    for (int i = front; i <= end; i++) {
        if (LeftSubArray[idxLeft] < RightSubArray[
            idxRight]) {
            Array[i] = LeftSubArray[idxLeft];
            idxLeft++;
        } else {
            Array[i] = RightSubArray[idxRight];
            idxRight++;
        }
    }
}

void MergeSort(vector<int> &Array, int front, int end)
    {
    if (front >= end)
        return;
    int mid = front + (end - front) / 2;
    MergeSort(Array, front, mid);
    MergeSort(Array, mid + 1, end);
    Merge(Array, front, mid, end);
}
```

4 Graph

4.1 Adjacency list for DFS And BFS

```
/*Adjacency list for DFS And BFS*/
#define N 205 //size

vector<int> adj[N]; //adjacency list
vector<bool> vis;   //visit

//DFS
void dfs(int x){
    vis[x]=1;
    for(int i:adj[x]){
        if(!vis[i])
            dfs(i);
    }
}

//BFS
void bfs(int s){
    queue<int> q;
    q.push(s);
    vis[s]=1;
    while(!q.empty()){
        int x=q.front();q.pop();
        for(int i:ADJ[x]){
            if(!vis[i])
                q.push(i),vis[i]=1;
        }
    }
}

void init(int N){
    for(int i=0;i<N;i++){
        if(!adj[i].empty()) adj[i].clear();
    }
}

int main() {

    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);

  return 0;
}
```

4.2 Disjoint Set(Union-Find)

```
/*Disjoint Set(Union-Find) 並查集*/
int f[N];    // 宣告父節點陣列 f
void init(int n) {
    for (int i = 0; i < n; i++)
        f[i] = i;
}
int find(int x) {
    return f[x] == x ? x : f[x] = find(f[x]);
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x != y)    f[y] = x;
}
```

4.3 Kruskal’s algorithm 最小生成樹

```
/*Kruskal’s algorithm 最小生成樹*/
//搭配 Disjoint Set(Union-Find)



struct Edge {
    int u, v, w;  // 點 u 連到點 v 並且邊權為 w
    friend bool operator<(const Edge& lhs, const Edge&
        rhs) {
        return lhs.w > rhs.w;//兩條邊比較大小用邊權比較
    }
};
```

```
priority_queue<Edge> graph();// 宣告邊型態的陣列 graph
int kruskal(int m){
    int tot = 0;
    for (int i = 0; i < m ; i++) {
        if (find(graph.top().u) != find(graph.top().v))
             { // 如果兩點未聯通
            merge(graph.top().u, graph.top().v);
               // 將兩點設成同一個集合
            tot += graph.top().w;  // 權重加進答案
        }
        graph.pop();
    }
    return tot;
}

int main() {
    int u, v, w, n, m,;
    cin >> n >> m; //node,edge
    init(n);
    for (int i = 0; i < m; i++) {
        cin >> u >> v >> w;
        graph.push(Edge{u,v,w});
    }
    cout << kruskal(m) << "\n";
    return 0;
}
```

4.4 Dijkstra’s Algorithm

```
/*Dijkstra's algorithm 單源最短路徑*/
#define MAX_V 100
#define INF 10000

struct Edge {
  int idx,w;
};
bool operator>(const Edge& a, const Edge& b) {
  return a.w > b.w;
}

int dist[MAX_V];
vector<vector<Edge> > adj(MAX_V);
void dijkstra(int vn, int s) {
  vector <bool> vis(vn, false);
  fill(dist, dist + vn, INF); dist[s] = 0;

  priority_queue <Edge, vector<Edge>, greater<Edge> >
      pq;
  Edge node;
  node.idx = s; node.w = 0;
  pq.emplace(node);
  while (!pq.empty()) {
    int u = pq.top().idx; pq.pop();
    if (vis[u])continue;
    vis[u] = true;
    for (auto v : adj[u]) {
      if (dist[v.idx] > dist[u] + v.w) {
        dist[v.idx] = dist[u] + v.w;
        node.w = dist[v.idx];
        node.idx = v.idx;
        pq.emplace(node);
      }

    }
  }
}

int main() {
    int start, end, u, v, w, i, n, m;
    cin >> n >> m; //node,edge
    for(i=0;i<m;i++){
        cin >> u >> v >> w;
        Edge node;
        node.idx = v; node.w = w;
        adj[u].push_back(node);
    }
    //從start連接到end的最短路徑
    cin >> start >> end;
    dijkstra(n, start);
    if(dist[end]==INF) cout << "NO\n";
    else cout << dist[end] <<"\n";
  return 0;
}
```

4.5 SPFA 單源最短路徑 (negative cycle)

```
/*SPFA 單源最短路徑(negative cycle)*/
struct Edge {
    int idx, w;
};
vector<Edge> adj[MAX_V]; //adjacency list
vector<bool> inp(MAX_V);
int dist[MAX_V];

//return true if negative cycle exists
bool spfa(int vn, int s) {
    fill(dist, dist + vn, INF); dist[s] = 0;
    vector<int> cnt(vn, 0);
    vector<bool> inq(vn, 0);
    queue<int> q; q.push(s); inq[s] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        inq[u] = false;
        for (auto v : adj[u]) {
            if (dist[v.idx] > dist[u] + v.w) {
                if (++cnt[v.idx] >= vn)return true;
                dist[v.idx] = dist[u] + v.w;
                if (!inq[v.idx]) inq[v.idx] = true, q.
                    push(v.idx);
            }
        }
    }
    return false;
}
```

4.6 Floyd-Warshall 全點對最短路徑

```
/*Floyd-Warshall 全點對最短路徑*/
//建立dp表，查詢任一點對最短路徑。
void floyd(){
    //將每個點對距離設為INF
    memset(dist,0x3f3f3f3f,sizeof(dist));
    //dist[u][v]為點u到點v的最短路徑
    //自己到自己的距離設為0
    for(int i=0;i<n;i++)  dist[i][i]=0;
    //輸入圖
    for(int i=0;i<m;i++)  cin>>u>>v>>w,dist[u][v]=w;
    for(int i=0;i<n;i++)     //窮舉中繼點
        for(int j=0;j<n;j++) //j,k窮舉點對
            for(int k=0;k<n;k++)
                dist[j][k]=min(dist[j][k],dist[j][i]+
                    dist[i][k]);
}
```

5 DP

```
/*如何設計DP?*/
// 設計狀態，先決定好要計算的東西(實際意義)與其參數
// 試著將任一狀態的答案用子狀態來表達(當然也要想清楚正
    確性)
// 列出轉移式(將2.的結果清楚寫下來)
// 確定其複雜度是否是好的
// DP優化(?)
```

5.1 背包問題

```
/*背包問題*/

// n：第0種到第n種物品要放進背包內。
// w：背包耐重限制。
// c(n, w)：只有第0種到第n種物品
// 耐重限制為w，此時的背包問題答案。
// weight[n]：第n種物品的重量。
// cost[n]：第n種物品的價值。
// number[n]：第n種物品的數量。

// 0/1背包滾動
// 每種物品只會放進背包零個或一個。
const int N = 500, W = 2000000; //N個物品,耐重W
int cost[N], weight[N];
int c[W + 1];
void knapsack(int n, int w)
{
  c[0] = 0;
  for (int i = 0; i < n; ++i)
    for (int j = w; j - weight[i] >= 0; --j)
      c[j] = max(c[j], c[j - weight[i]] + cost[i]);
  cout << c[w];
}
// 0/1背包可用於：
// 一個數字集合，挑幾個數字，總和恰為零（Subset Sum
     Problem)
// 一個數字集合，挑幾個數字，總和恰為整體總和的一半（
     Partition Problem)
// N個不同重量物品，M個不同耐重箱子，用最少箱子裝所有物
     品（Bin Packing Problem)

// 無限背包
// 物品有許多種類，每一種物品都無限量供應的背包問題。
void knapsack(int n, int w)
{
    memset(c, 0, sizeof(c));
    for (int i=0; i<n; ++i)
        for (int j = weight[i]; j <= w; ++j)
            c[j] = max(c[j], c[j - weight[i]] + cost[i
                ]);

    cout << "最高的價值為" << c[w];
}

// 有限背包
// 物品有許多種類，每一種物品都是限量供應的背包問題。
int cost[N], weight[N], number[N];
// number[n]：第n種物品的數量。
void knapsack(int n, int w)
{
    for (int i = 0; i < n; ++i)
    {
        int num = min(number[i], w / weight[i]);
        for (int k = 1; num > 0; k *= 2)
        {
            if (k > num) k = num;
            num -= k;
            for (int j = w; j >= weight[i] * k; --j)
                c[j] = max(c[j], c[j - weight[i] * k] +
                     cost[i] * k);
        }
    }
    cout << "最高的價值為" << c[w];
}
```

5.2 找錢問題

```
/*Money Changing Problem*/
// n：用第0種到第n種錢幣來湊得價位。
// m：欲湊得的價位值。
// c(n, m)：用第0種到第n種錢幣湊得價位m的湊法數目。
// price[n]：第n種錢幣的面額大小。

// 能否湊得某個價位（ Money Changing Problem ）
// 給定許多種不同面額的錢幣，
// 能否湊得某個價位？
// 每種面額的錢幣都無限供應。

// 錢幣面額，順序可隨意。
int price[5] = {5, 2, 6, 11, 17};
bool c[1000+1];

// 看看 {5, 2, 6, 11, 17} 這些面額湊不湊得到價位 m
void change(int m)
{
    memset(c, false, sizeof(c));
    c[0] = true;

    // 依序加入各種面額
    for (int i = 0; i < 5; ++i)
        // 由低價位逐步到高價位
        for (int j = price[i]; j <= m; ++j)
            // 湊、湊、湊
            c[j] |= c[j-price[i]];

    if (c[m])
        cout << "湊得到";
    else
        cout << "湊不到";
}

// 湊得某個價位的湊法總共幾種（ Coin Change Problem ）
void change(int m)
{
    memset(c, 0, sizeof(c));
    c[0] = 1;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            c[j] += c[j-price[i]];

    cout << "湊得價位" << m;
    cout << "湊法總共" << c[m] << "種";
}

// 湊得某個價位的最少錢幣用量（ Change-Making Problem
    ）
// c(n, m)：用第0種到第n種錢幣湊得價位m，最少所需要的錢
    幣數量。
void change(int m)
{
    memset(c, 0x7f, sizeof(c));
    c[0] = 0;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            c[j] = min(c[j], c[j-price[i]] + 1);

    cout << "湊得價位" << m;
    cout << "最少需（只）要" << c[m] << "個錢幣";
}

// 湊得某個價位的錢幣用量，有哪幾種可能性。
void change(int m)
{
    memset(c, 0, sizeof(c));
    c[0] = 1;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            // 錢幣數量加一，每一種可能性都加一。
            c[j] |= c[j-price[i]] << 1;

    for (int i = 1; i <= 63; ++i)
        if (c[m] & (1 << i))
```

```
            cout << "用" << i << "個錢幣可湊得價位" <<
                m;
}

// 能否湊得某個價位，但是錢幣限量供應！
int price[5] = {5, 2, 6, 11, 17};
int number[5] = {4, 5, 5, 3, 2};    // 各種錢幣的供應數
    量
bool c[1000+1];

void change(int m)
{
    memset(c, 0, sizeof(c));
    c[0] = true;

    for (int i = 0; i < 5; ++i)
        // 各種餘數分開處理
        for (int k = 0; k < price[i]; ++k)
        {
            int left = number[i];   // 補充彈藥

            // 由低價位到高價位
            for (int j = k; j <= m; j += price[i])
                // 先前的面額已能湊得，當前面額可以省著
                    用。
                if (c[j])
                    left = number[i];   // 補充彈藥

                // 過去都無法湊得，一定要用目前面額硬
                    湊。
                else if (left > 0)
                {
                    left--; // 用掉一個錢幣
                    c[j] = true;
                }
        }

    if (c[m])
        cout << "湊得到";
    else
        cout << "湊不到";
}

// Cashier's Algorithm
// 買東西找回最少硬幣。
int price[5] = {50, 20, 10, 4, 2};  // 面額由大到小排列

void cashier(int n) // n 是總共要找的錢。
{
    int c = 0;
    for (int i=0; i<5; ++i)
        while (n >= price[i])
        {
            n -= price[i];  // 找了 price[i] 元
            c++;
        }

    if (n != 0)
        cout << "找不出來";
    else
        cout << "找了" << c << "個錢幣";
}
```

5.3 最長公共子序列 LCS

```
/*LCS 最長公共子序列*/
void LCS() {
    for (int i = 0; i <= n1; i++) length[i][0] = 0;
    for (int j = 0; j <= n2; j++) length[0][j] = 0;
    for (int i = 1; i <= n1; i++)
        for (int j = 1; j <= n2; j++)
            if (s1[i] == s2[j]) {
                length[i][j] = length[i - 1][j - 1] +
                    1;
                prev[i][j] = 0;  // 左上方
            }
            else {
                if (length[i - 1][j] < length[i][j -
                    1]) {
                    length[i][j] = length[i][j - 1];
                    prev[i][j] = 1;  // 左方
                }
                else {
                    length[i][j] = length[i - 1][j];
                    prev[i][j] = 2;  // 上方
                }
            }
    cout << "LCS的長度是" << length[n1][n2];
    cout << "LCS是";
    print_LCS(n1, n2);
}
void print_LCS(int i, int j) {
    if (i == 0 || j == 0) return;
    if (prev[i][j] == 0) {
        print_LCS(i - 1, j - 1);
        cout << s1[i];              // 印出LCS的元素
    }
    else if (prev[i][j] == 1)  // 左方
        print_LCS(i, j - 1);
    else if (prev[i][j] == 2)  // 上方
        print_LCS(i - 1, j);
}
```

5.4 最長遞增子序列 LIS

```
/*LIS 最長遞增子序列*/
void LIS() {
  for (int i = 0; i < n; i++) length[i] = 1;
  for (int j = 0; j < n; j++) {
    for (int i = j + 1; i < n; i++)
      if (s[j] < s[i])length[i] = max(length[i], length
          [j] + 1);
  }

  int l = 0;
  for (int i = 0; i < n; i++) {
    l = max(length[i], l);
  }
  cout << l;
}
```

5.5 最大非連續子序列和

```
/*最大非連續子序列和*/
int sub_max(int* list,int sub_len) { //子序列長度
    sub_len
    if (sub_len == 3) {
        return list[0] + list[2];
    }
    int temp[10005];
    for (int m = 0; m < sub_len; m++) {
        temp[m] = list[m];
    }
    temp[0] = list[0];
    temp[1] = list[1] > list[0] ? list[1] : list[0];
    for (int i = 2; i < sub_len; i++) {
        temp[i] = max(max(temp[i], temp[i - 1]), temp[i
             - 2] + list[i]);
    }
    return temp[sub_len - 1];
}
int main() {

    int n, m;
    int list[10005];

    cin >> n;
    for (m = 0; m < n; m++) {
        cin >> list[m];
    }
    sub_len = m;//list大小,global變數

    cout << sub_max(list, sub_len);

    return 0;
}
```

6 STL tool

6.1 常用工具

```
/*--------常用工具--------*/
swap(a,b);
min(a,b);
max({ a, b, c });

//二進制"1"的個數
__builtin_popcount(n)   -> int
__builtin_popcountl(n)  -> long int
__builtin_popcountll(n) -> long long

//math
abs(x);
pow(x);
sqrt(x);
__gcd(x, y);
__lg(x)  //以2為底數
log(x)   //以e為底數
log10(x) //以10為底數
do {     //排列組合
    cout << s << "\n";
} while (next_permutation(s.begin(), s.end()));


//陣列處理
sort(arr,arr+n);
reverse(arr,arr+n);
*min_element(arr, arr+n); //value
min_element(arr, arr+n) - arr; //index
*lower_bound(arr, arr+4, c) << '\n'; //第一個大於等於c
*upper_bound(arr, arr+4, c) << '\n'; //第一個大於c
//填充 arr[0]=123 arr[1]=123 arr[2]=123
fill(arr, arr+3, 123);

//輸出
//四捨五入 或是更高精度(int)10 * 位數 + 0.5
cout << fixed << setprecision(10);
//寬度n 用char(c)填補
cout << setw(n) << setfill(c) << ;

//迭代器
T.begin()
T.end()
T.rbegin() //逆序迭代器
T.rend() //逆序迭代器
T.find() //可用於set,map的earse()。
```

6.2 Sort

```
/*--------sort--------*/

//cmp
struct T {int val, num;};
bool cmp(const T &a, const T &b) {
    return a.num < b.num;
}
sort(arr.begin(), arr.end(), cmp);

//operator
struct Point {
    int x, y;
    bool operator<(Point b) {
        if (x != b.x) return x < b.x;
        else return y < b.y;
    }
};
Point arr[n];
sort(arr, arr+n); //二維平面，從小到大排列。
```

6.3 Stack

```
/*--------stack--------*/
• push()
• pop()
• top()
• empty()
• size()
```

6.4 Queuet

```
/*--------queue--------*/
• push()
• pop()
• front()
• empty()
• size()
```

6.5 Priority Queue

```
/*--------priority_queue--------*/
• top()
• push()
• pop()
• emplace()

//預設由大排到小
priority_queue<T> pq
priority_queue<int, vector<int>, less<int> > pq;
//改成由小排到大
priority_queue<T, vector<T>, greater<T> > pq;
//自行定義 cmp 排序
priority_queue<T, vector<T>, cmp> pq;


struct cmp {
    bool operator()(node a, node b) {
        //priority_queue優先判定為!cmp
  //，所以「由大排到小」需「反向」定義
        //實現「最小值優先」
        return a.x < b.x;
    }
};
```

6.6 List

```
/*--------list--------*/
• push_back()
• pop_back()
• push_front()
• pop_front()
• back()
• front()
• insert(index, obj)
• erase()
```

```
//遍歷
for (auto iter = _list.begin(); iter != _list.end();
    iter++)
    cout << *iter << "\n";
```

6.7 Set

```
/*--------set--------*/
• insert()
• erase(l, r) //l與r皆為iterator
• erase()
• empty()
• clear()
• count() //元素是否存在

//遍歷
int mints[] = { 75,23,65,42,13,75,65 };
set<int> myset(myints, myints + 7);
for (auto it = myset.begin(); it != myset.end(); it++)
    cout << ' ' << *it;
```

6.8 Map

```
/*--------map--------*/
map<char, int> mymap;
mymap['b'] = 100, mymap['a'] = 200, mymap['c'] = 300;

//find
auto iter = mymap.find("a");
if (iter != mapStudent.end())
    cout << "Find, the value is" << iter->second <<
        endl;
else
    cout << "Do not Find" << endl;

//erase
auto iter = mymap.find("a");
mymap.erase(iter);

//map遍歷
for (auto it = mymap.begin(); it != mymap.end(); it++)
cout << it->first << ", " << it->second << endl
```

6.9 Stringstream

```
/*--------stringstream--------*/
stringstream ss;
• getline(cin, str);
• ss.str("");
• ss.clear();

//實現"切割"以及"型態轉換"
//int_to_string
ss << n;
ss >> str;

//string_to_int
ss << str;
ss >> n;

//注意輸入時，cin後的快取問題
cin >> n;
getline(cin, str);  //str = endl
getline(cin, str);  //str = 目標str

//實現"進制轉換"
ss << oct << s;    //以8進制讀入流中
ss << hex << s;    //以16進制讀入流中
ss >> n;        //10進制int型輸出
ss >> s;        //x進制str型輸出
```

6.10 Bitset

```
/*--------bitset--------*/
//init
string s = "1001101";
bitset<10> b(s);

b.set();    //每個位元設 '1'
b.reset();  //每個位元設 '0'
b[pos] = 1;

//轉換
s = b.to_string();
unsigned long x = b.to_ulong();

//overload
b = !b0;
b = b0 & b1;
b = b0 | b1;
b = b0 ^ b1;

//shift
new_b = b << 2;
new_b = b >> 2;

//sum
b.any();//判別是否有 '1'
b.none();//判別是否沒 '1'
cnt = b.count();// 判別 '1' 之個數
cnt = b.size() - b.count();//判別 '0' 之個數
```

7 Other

7.1 Basic

```
/*前置作業*/
#include <bits/stdc++.h>
#define ll long long
#define ld long double
using namespace std;
int main() {
    cin.tie(0); //取消強制flush
    ios_base::sync_with_stdio(false); //取消 iostream
        與 stdio 的同步使用
}

/*unroll-loops*/
#pragma GCC optimize("O0")//不優化(預設)
#pragma GCC optimize("O1")//優化一點
#pragma GCC optimize("O2")//優化更多
#pragma GCC optimize("O3")//O2優化再加上inline函式優化
#pragma GCC optimize("unroll-loops")

/*常數宣告*/
// 數字中可以加 ' 方便看出幾位數
#define MXN 1'000'005
// 1e-6 為科學記號 代表 1 * 10^-6
#define EPS 1e-6
// 0x3f3f3f3f為一個接近10^9的數字0x為16進位
#define INF 0x3f3f3f3f
// acos(-1) 等同圓周率
#define PI acos(-1)

/*位元運算*/
if(x&1) cout<<奇數;
else    cout<<偶數;
x <<= 1   //將x左移1，等同 *2
x >>= 2   //將x右移2，等同 /4

/*include <bits/stdc++.h>
C:\Program Files\Microsoft Visual
```

```
Studio\2022\Community\VC\Tools\MSVC\14.30.30705\include
    \bits*/
```

7.2 Header

```
// C
#ifndef _GLIBCXX_NO_ASSERT
#include <cassert>
#endif
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <ciso646>
#include <climits>
#include <clocale>
#include <cmath>
#include <csetjmp>
#include <csignal>
#include <cstdarg>
#include <cstddef>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>

#if __cplusplus >= 201103L
#include <ccomplex>
#include <cfenv>
#include <cinttypes>
#include <cstdalign>
#include <cstdbool>
#include <cstdint>
#include <ctgmath>
#include <cwchar>
#include <cwctype>
#endif

// C++
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <new>
#include <numeric>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>

#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
#include <initializer_list>
#include <mutex>
#include <random>
#include <ratio>
#include <regex>
#include <scoped_allocator>
#include <system_error>
#include <thread>
#include <tuple>
#include <typeindex>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#endif
```

```
                                  @
               (__)    (__) ____/ Hong~Long~Long~Long~
            /| (oo) _  (oo)/----/_____    *
  _o\______/_|\_\/_/_|__\/|____|//////== *- *  * -
 /__________  \   AC |  AC | NO BUG /== -*
[_____/^^\_____\_____|_____/^^\_____]     *- * -
      \__/                 \__/      Chong~Chong~Chong~
```