Contents

1 Data Structure

1.1 Binary Search

```
int binary_search(int arr[maxn], int lef, int rig, int target){
    if(lef > rig) return 0x3f3f3f3f;
    int mid = (lef + rig) >> 1;
    if(arr[mid] == target) return mid;
    else if(arr[mid] > target){
        return binary_search(arr, lef, mid - 1, target);
    }
    else{
        return binary_search(arr, mid + 1, rig, target);
    }
}
```

1.2 BIT

```
#define lowbit(k) (k & -k)
void add(vector<int> &tr, int id, int val) {
  for (; id <= n; id += lowbit(id)) {
    tr[id] += val;
  }
}
int sum(vector<int> &tr, int id) {
  int ret = 0;
  for (; id >= 1; id -= lowbit(id)) {
    ret += tr[id];
  }
  return ret;
}
```

1.3 Segment tree

```
int dfs(int lef, int rig){
    if(lef + 2 == rig){
        if(num[lef] > num[rig-1]){
            return lef;
        }
        else{
            return rig-1;
        }
    }
    int mid = (lef + rig)/2;
    int p1 = dfs(lef, mid);
    int p2 = dfs(mid, rig);
    if(num[p1] > num[p2]){
        return p1;
    }
    else{
        return p2;
    }
}
```

1.4 Trie

```
const int MAXL = ; // 自己填
const int MAXC = ;
struct Trie {
  int nex[MAXL][MAXC];
  int len[MAXL];
  int sz;
  void init() {
    memset(nex, 0, sizeof(nex));
    memset(len, 0, sizeof(len));
    sz = 0;
  }
  void insert(const string &str) {
    int p = 0;
    for (char c : str) {
      int id = c - 'a';
      if (!nex[p][id]) {
        nex[p][id] = ++sz;
      }
      p = nex[p][id];
    }
    len[p] = str.length();
  }
  vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
      int id = str[i] - 'a';
      if (!nex[p][id]) {
        return ans;
      }
      p = nex[p][id];
      if (len[p]) {
        ans.pb(len[p]);
      }
    }
    return ans;
  }
};
```

1.5 BWT

```
/*BWT 資料轉換演算法*/
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0)
            mini[zero++] = i;
    for(int i = 0; i < n; ++i)
        if(back[i] == 1)
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
```

2 Divide and Conquer

2.1 count inversions

```
/*逆序數對*/
int arr[maxn], buf[maxn];
int count_inversions(int lef, int rig){
    if(rig - lef <= 1) return 0;
    int mid = (lef + rig)/2;
    int ans = count_inversions(lef, mid) +
        count_inversions(mid, rig);
    int i = lef, j = mid, k = lef;
    while(i < mid || j < rig){
        if(i >= mid) buf[k] = arr[j++];
        else if(j >= rig) buf[k] = arr[i++];
        else{
            if(arr[i] <= arr[j]) buf[k] = arr[i++];
            else{
                buf[k] = arr[j++];
                ans += mid - i;
            }
        }
        k++;
    }
    for(int k = lef; k < rig; ++k) arr[k] = buf[k];
    return ans;
}
```

3 DP

3.1 Doubling

```
/* 倍增 */
int LOG = sqrt(N); // 2^LOG >= N
vector<int> arr(N);
vector<vector<int>> dp(N, vector<int>(LOG));
for(int i = 0; i < N; ++i) cin >> arr[i];
int L, Q, a, b;
cin >> L >> Q;
for(int i = 0; i < N; ++i){
    dp[i][0] = lower_bound(arr.begin(), arr.end(),
        arr[i] + L) - arr.begin();
    if(dp[i][0] == N || arr[i] + L < arr[dp[i][0]])
        dp[i][0] -= 1;
}
for(int i = 1; i < LOG; ++i)
    for(int j = 0; j < N; ++j)
        dp[j][i] = dp[dp[j][i - 1]][i - 1];
for(int i = 0; i < Q; ++i){
    cin >> a >> b;
    a--; // 要減減是因為arr的index從0開始但題目從1開始
    b--;
    if(a > b) swap(a, b);
    int ans = 0;
    for(int i = LOG - 1; i >= 0; --i){ // 從後往回推
        if(dp[a][i] < b){
            ans += (1 << i);
            a = dp[a][i];
        }
    }
    cout << ans + 1 << endl;
}
```

3.2 Josephus

```
int josephus (int n, int k) {
    // 有 n 個人圍成一圈，每 k 個一次
    return n > 1 ? (josephus(n-1 , k) + k) % n : 0;
}
// 回傳最後一人的編號, 0 index
```

3.3 LCS

```
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  int dp[n1+1][n2+1] = {0};
  // dp[i][j] = s1的前i個字元和s2的前j個字元
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[n1][n2];
}
```

3.4 LIS

```
int LIS(vector<int> &a) { // Longest Increasing Subsequence
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

4 Enumerate

4.1 Halfcut Enumerate

```
/* 折半枚舉 */
void dfs(set<long long int> &s, int depth, int T,
    long long int sum){
    if(depth >= T){
        s.insert(sum);
        return;
    }
    dfs(s, depth + 1, T, sum); // 取或不取的概念
    dfs(s, depth + 1, T, sum + A[depth]);
}
int main(){
    int N, T;
    set<long long int> s1, s2;
    cin >> N >> T;
    for(int i = 0; i < N; ++i) cin >> A[i];
    dfs(s1, 0, N/2, 0); // 折半枚舉
    dfs(s2, N/2, N, 0);
    long long int ans = 0;
    // 題目:枚舉集合 Sx 的數字 Sxi，找出 Sy
        集合內小於等於 T-Sxi 中最大的數 Syj
    for(auto &x : s1){
        auto it = s2.upper_bound(T - x);
        long long int y = *(--it);
        if(x + y <= T) ans = max(ans, x + y);
    }
    cout << ans << endl;
}
```

5 Graph

5.1 SPFA

```
bool SPFA(int s){
    // 記得初始化這些陣列
    int cnt[1000+5], dis[1000+5];
    bool inqueue[1000+5];
    queue<int> q;

    q.push(s);
    dis[s] = 0;
    inqueue[s] = true;
    cnt[s] = 1;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        inqueue[now] = false;

        for(auto &e : G[now]){
            if(dis[e.t] > dis[now] + e.w){
                dis[e.t] = dis[now] + e.w;
                if(!inqueue[e.t]){
                    cnt[e.t]++;
                    if(cnt[e.t] > m){
                        return false;
                    }
                    inqueue[e.t] = true;
                    q.push(e.t);
                }
            }
        }
    }
    return true;
}
```

5.2 Dijkstra

```
struct Item{
    int u, dis;
    // 取路徑最短
    bool operator < (const Item &other) const{
        return dis > other.dis;
    }
};
int dis[maxn];
vector<Edge> G[maxn];
void dijkstra(int s){
    for(int i = 0; i <= n; i++){
        dis[i] = inf;
    }
    dis[s] = 0;
    priority_queue<Item> pq;
    pq.push({s, 0});
    while(!pq.empty()){
        // 取路徑最短的點
        Item now = pq.top();
        pq.pop();
        if(now.dis > dis[now.u]){
            continue;
        }
        // 鬆弛更新，把與 now.u 相連的點都跑一遍
        for(Edge e : G[now.u]){
            if(dis[e.v] > now.dis + e.w){
                dis[e.v] = now.dis + e.w;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
```

5.3 Floyd Warshall

```
void floyd_warshall(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            G[i][j] = INF;
        }
        G[i][i] = 0;
    }
    for (int k = 0; k < n; k++){      //
        嘗試每一個中繼點
        for (int i = 0; i < n; i++){ //
            計算每一個i點與每一個j點
            for (int j = 0; j < n; j++){
                G[i][j] = min(G[i][j], G[i][k] +
                    G[k][j]);
            }
        }
    }
}
```

5.4 Disjoint set Kruskal

```
struct Edge{
    int u, v, w;
    // 用權重排序 由大到小
    bool operator < (const Edge &other) const{
        return w > other.w;
    }
}edge[maxn];
// disjoint set
int find(int x){
  if(parent[x] < 0){
    return x;
  }
  else{
    return parent[x] = find(parent[x]);
  }
}
void unite(int a, int b){
  a = find(a);
  b = find(b);

  if(a != b){
    if(parent[a] < parent[b]){
      parent[a] += parent[b];
      parent[b] = a;
    }
    else{
      parent[b] += parent[a];
      parent[a] = b;
    }
  }
}
void kruskal(){
    memset(parent, -1, sizeof(parent));
    sort(edge, edge + m);
    int i, j;
    for(i = 0, j = 0; i < n - 1 && j < m; i++){
        // 如果 u 和 v 的祖先相同, 則 j++ (祖先相同代表會產生環 所以不要)
        while(find(edge[j].u) == find(edge[j].v)) j++;
        // 若部會產生環 則讓兩點之間產生橋 (連接兩顆子生成樹)
        unite(edge[j].u, edge[j].v);
        j++;
    }
}
```

5.5 KM

```
const int X = 50;    // X的點數，等於Y的點數
const int Y = 50;    // Y的點數
int adj[X][Y];       // 精簡過的adjacency matrix
int lx[X], ly[Y];    // vertex labeling
int mx[X], my[Y];    // X各點的配對對象、Y各點的配對對象
int q[X], *qf, *qb; // BFS queue
int p[X];            // BFS parent，交錯樹之偶點，指向上一個偶點
bool vx[X], vy[Y];   // 記錄是否在交錯樹上
int dy[Y], pdy[Y];   // 表格

void relax(int x){ // relaxation
    for (int y=0; y<Y; ++y)
        if (adj[x][y] != 1e9)
            if (lx[x] + ly[y] - adj[x][y] < dy[y]){
                dy[y] = lx[x] + ly[y] - adj[x][y];
                pdy[y] = x; // 記錄好是從哪個樹葉連出去的
            }
}
void reweight(){ // 調整權重、調整表格
    int d = 1e9;
    for (int y=0; y<Y; ++y) if (!vy[y]) d = min(d, dy[y]);
    for (int x=0; x<X; ++x) if ( vx[x]) lx[x] -= d;
    for (int y=0; y<Y; ++y) if ( vy[y]) ly[y] += d;
    for (int y=0; y<Y; ++y) if (!vy[y]) dy[y] -= d;
}
void augment(int x, int y){ // 擴充路徑
    for (int ty; x != -1; x = p[x], y = ty){
        ty = mx[x]; my[y] = x; mx[x] = y;
    }
}
bool branch1(){ // 延展交錯樹：使用既有的等邊
    while (qf < qb)
        for (int x=*qf++, y=0; y<Y; ++y)
            if (!vy[y] && lx[x] + ly[y] == adj[x][y]){
                vy[y] = true;
                if (my[y] == -1){
                    augment(x, y);
                    return true;
                }
                int z = my[y];
                *qb++ = z; p[z] = x; vx[z] = true; relax(z);
            }
    return false;
}
bool branch2(){ // 延展交錯樹：使用新添的等邊
    for (int y=0; y<Y; ++y){
        if (!vy[y] && dy[y] == 0){
            vy[y] = true;
            if (my[y] == -1){
                augment(pdy[y], y);
                return true;
            }
            int z = my[y];
            *qb++ = z; p[z] = pdy[y]; vx[z] = true; relax(z);
        }
    }
    return false;
}
int Hungarian(){
    // 初始化vertex labeling
    //  memset(lx, 0, sizeof(lx));  // 任意值皆可
    memset(ly, 0, sizeof(ly));
    for (int x=0; x<X; ++x)
        for (int y=0; y<Y; ++y)
            lx[x] = max(lx[x], adj[x][y]);

    // X側每一個點，分別建立等邊交錯樹。
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    for (int x=0; x<X; ++x){
        memset(vx, false, sizeof(vx));
        memset(vy, false, sizeof(vy));
        memset(dy, 0x7f, sizeof(dy));
        qf = qb = q;
        *qb++ = x; p[x] = -1; vx[x] = true; relax(x);
        while (true){
            if (branch1()) break;
            reweight();
            if (branch2()) break;
        }
    }
    // 計算最大權完美匹配的權重
    int weight = 0;
    for (int x=0; x<X; ++x)
        weight += adj[x][mx[x]];
    return weight;
}
```

5.6 Dinic

```
// Maximum Flow
const int V = 100, E = 1000;
int adj[V]; // adjacency lists，初始化為-1。
struct Element {int b, r, next;} e[E*2];
int en = 0;
void addedge(int a, int b, int c){
    e[en] = (Element){b, c, adj[a]}; adj[a] = en++;
    e[en] = (Element){a, 0, adj[b]}; adj[b] = en++;
}
int d[V];       // 最短距離
bool visit[V];  // BFS/DFS visit record
int q[V];       // queue
int BFS(int s, int t){ // 計算最短路徑，求出容許圖
    memset(d, 0x7f, sizeof(d));
    memset(visit, false, sizeof(visit));
    int qn = 0;
    d[s] = 0;
    visit[s] = true;
    q[qn++] = s;

    for (int qf=0; qf<qn; ++qf){
        int a = q[qf];
        for (int i = adj[a]; i != -1; i = e[i].next){
            int b = e[i].b;
            if (e[i].r > 0 && !visit[b]){
                d[b] = d[a] + 1;
                visit[b] = true;
                q[qn++] = b;
                if (b == t) return d[t];
            }
        }
    }
    return V;
}
int DFS(int a, int df, int s, int t){ // 求出一條最短擴充路徑，並擴充流量
    if (a == t) return df;
    if (visit[a]) return 0;
    visit[a] = true;
    for (int i = adj[a]; i != -1; i = e[i].next){
        int b = e[i].b;
        if (e[i].r > 0 && d[a] + 1 == d[b]){
            int f = DFS(b, min(df, e[i].r), s, t);
            if (f){
                e[i].r -= f;
                e[i^1].r += f;
                return f;
            }
        }
    }
    return 0;
}
int dinitz(int s, int t){
    int flow = 0;
    while (BFS(s, t) < V)
        while (true){
            memset(visit, false, sizeof(visit));
            int f = DFS(s, 1e9, s, t);
            if (!f) break;
            flow += f;
        }
    return flow;
}
```

5.7 Bipatirate

```
const int maxn = 300 + 5;
int n, color[maxn];
vector<vector<int>> v(maxn);
bool dfs(int s){
    for(auto it : v[s]){
        if(color[it] == -1){
            color[it] = 3 - color[s];
            if(!dfs(it)){
                return false;
            }
        }
        if(color[s] == color[it]){
            return false;
        }
    }
    return true;
}
void isBipatirate(){
    bool flag = true;
    for(int i = 1; i <= n; ++i){
        if(color[i] == -1){
            color[i] = 1;
            flag &= dfs(i);
        }
    }
    if(flag){
        cout << "YES" << endl;
    }
    else{
        cout << "NO" << endl;
    }
}
int main(){
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) v[i].clear();
        memset(color, -1, sizeof(color));
        int a, b;
        while(cin >> a >> b && (a || b)){
            v[a].emplace_back(b);
            v[b].emplace_back(a);
        }
        isBipatirate();
    }
}
```

5.8 Hungarian algorithm

```
const int maxn = 500+5;
int t, N, bn, gn, match[maxn];
bool visited[maxn];
vector<vector<int>> G(maxn);
struct People{
    int h;
    string music, sport;
    People(){}
    People(int h, string music, string sport){
        this->h = h;
        this->music = music;
        this->sport = sport;
    }
}lef[maxn], rig[maxn];
bool check(People boy, People girl){
    if(abs(boy.h - girl.h) <= 40 && boy.music == girl.music && boy.sport != girl.sport) return true;
    return false;
}
bool dfs(int s){
    for(int i = 0; i < G[s].size(); ++i){
        int v = G[s][i];
        if(visited[v]) continue;
        visited[v] = true;
        if(match[v] == -1 || dfs(match[v])){
            match[v] = s;
            return true;
        }
    }
    return false;
}
int Hungarian(){
    int cnt = 0;
    memset(match, -1, sizeof(match));
    for(int i = 0; i < bn; ++i){
        memset(visited, false, sizeof(visited));
        if(dfs(i)) cnt++;
    }
    return cnt;
}
int main(){
    cin >> t;
    while(t--){
        cin >> N;
        bn = 0, gn = 0;
        for(int i = 0; i <= N; ++i) G[i].clear();
        int h;
        string sex, music, sport;
        for(int i = 0; i < N; ++i){
            cin >> h >> sex >> music >> sport;
            if(sex == "M") lef[bn++] = People(h, music, sport);
            else rig[gn++] = People(h, music, sport);
        }
        for(int i = 0; i < bn; ++i){
            for(int j = 0; j < gn; ++j)
                if(check(lef[i], rig[j]))
                    G[i].emplace_back(j);
        }
        cout << N - Hungarian() << endl;
    }
}
```

5.9 LCA

```
/*最低共同祖先*/
// 此 node 下有機顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] =
                parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA（利用倍增法）
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
```

6 Other

6.1 Ants Colony

```
/* LCA 最低共同祖先 */
const int maxn = 1e5 + 5;
struct Edge{
    int v;
    int w;
};
int N;
vector<Edge> G[maxn];
int parent[maxn][20+5];
int depth[maxn], siz[maxn];
// 此 node 下有機顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] =
                parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA（利用倍增法）
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
long long int dist[maxn];
// 從 0 開始到每個點的距離
void distance(){
    for (int u = 0; u < N; ++u){
        for(int i = 0; i < G[u].size(); ++i){
            dist[G[u][i].v] = dist[u] + G[u][i].w;
}
int main(){
    while(cin >> N && N){
        memset(dist, 0, sizeof(dist));
        memset(parent, 0, sizeof(parent));
        memset(depth, 0, sizeof(depth));
        memset(siz, 0, sizeof(siz));
        for(int i = 0; i <= N; ++i){
            G[i].clear();
        }
        for(int i = 1; i < N; ++i){
            int u, w;
            cin >> u >> w;
            G[u].push_back({i, w});
            parent[i][0] = u;
        }
        find_parent();
        dfs(0, 0);
        distance();
        int s; cin >> s;
        bool space = false;
        for(int i = 0; i < s; ++i){
            int a, b;
            cin >> a >> b;
            int lca = LCA(a, b);
            if(space) cout << " ";
            space = true;
            cout << (dist[a] + dist[b]) - (dist[lca]
                * 2);
        }
        cout << endl;
    }
}
```

6.2 Binary codes

```
/* BWT 資料轉換演算法 */
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0){
            mini[zero++] = i;
    for(int i = 0; i < n; ++i){
        if(back[i] == 1){
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
int main(){
    cin >> n;
    for(int i = 0; i < n; ++i){
        cin >> back[i];
    zero = 0;
    BWT();
}
```

7 Function

7.1 strstr

```
#include <stdio.h>
#include <string.h>

int main(){
char * c;
char str1[1005], str2[1005];
scanf("%s %s", str1, str2);
c = strstr(str1, str2);
if (c != NULL){
    printf("Yes\n");
}
else printf("No\n");
}
// Input : Hello eLl
// Output : No
```

7.2 substr

```
int main(){
    string str; //abcdef
    cin >> str;
    string tmp;
    tmp = str.substr(0, 2); //ab
    str = str.substr(2); //cdef
    cout << tmp << " " << str;
    return 0;
}
```

7.3 map set

```
.begin( ) // Return iterator to beginning
.end( ) // Return iterator to end
.empty( ) // 檢查是否為空
.size( )  // 回傳大小
mp.insert(pair<char,int>('a',100))
st.insert(100) // 插入key、value
.erase( ) // 刪掉指定key和他的value
.clear( ) // 清空整個 map
m.find( )
cout << "a => " << mymap.find('a')->second << endl;
    // 找出 map 裡 key
        有沒有在裡面，如果有的話會回傳元素所在的iterator，否則傳回隊尾
s.count() // 返回某個值元素在set的個數
while( !mymap.empty()){
    cout << mymap.begin()->first << " => " <<
        mymap.begin()->second << endl;
    mymap.erase(mymap.begin());
}
for (auto it = mymap.begin(); it != mymap.end(); ++it)
    cout << it->first << " => " << it->second << endl;
```

7.4 vector

```
v.erase(v.begin() + 5) //拿掉第六個數
v.erase (v.begin(), v.begin() + 3); //拿掉前三個數
```

7.5 setprecision

```
// 將數字的小數部分設定為固定長度
cnt = 3.5555;
cout << fixed << setprecision(3) << cnt ;
// output : 3.555
```

7.6 GCD LCM

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

/* 輾轉相除法 - 求兩數是否互質
如果兩數互質 最終結果其中一方為0時 另一方必為1
若兩數有公因數 最終結果其中一方為0時 另一方必不為1 */
while ( ( num1 %= num2 ) != 0 && ( num2 %= num1 ) !=
    0 );
```

7.7 reverse

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
reverse(a, a+5) // 轉換0~5

vector<int> v;
reverse(v.begin(), v.end());

string str = "123";
reverse(str.begin(), str.end());
cout << str << endl; //321
```

7.8 CHAR

```
isdigit()
isalnum() //判斷字母 || 數字
isalpha()
islower()
isupper()
isblank() //判斷是否為空格，或者 tab 健制表符，即
    space 和 \t
toupper()
tolower()
```

7.9 sort

```
priority_queue<int, vector<int>, less<int>> //大到小
priority_queue<int, vector<int>, greater<int>>
    //小到大

int arr[] = {4, 5, 8, 3, 7, 1, 2, 6, 10, 9};
    sort(arr, arr+10);

vector<int> v;
sort(v.begin(), v.end()); //小到大

int cmp(int a, int b){
    return a > b;
}
sort(v.begin(), v.end(), cmp); //大到小
```

7.10 struct

```cpp
struct area{
    int a, b;
    bool operator<(const area rhs) const{
        return a > rhs.a || ( a == a && b > rhs.b);
    }
    bool operator!=(const area rhs) const{
        return a != rhs.a || b != rhs.b;
    }
};
```

7.11 deque

```cpp
deque <int> que;
que.push_back(10);
que.push_front(20);
que.front()
que.back()
que.pop_front()
que.pop_back()
cout << "Element at position 2 : " << que.at(2) << endl;
```

7.12 python template

```python
import math
import operator

try:
    while(1):
        listx = []
        listx.append("...")
        list_s = sorted(listx) # 小到大
        list_s = sorted(listx, reverse = True) # 大到小
        # max(listx)
        # min(listx)
        # sum(listx)
        # len(listx)
        dicty = {}
        dicty[key] = "value"
        dicty= sorted(dicty.items()) # by key
        dicty= sorted(dicty.items(), key=operator.itemgetter(1)) # by value
        # EOF寫法
        # 階層 math.factorial(3) == 6
        # 絕對值 math.fabs(x)
        # 無條件進位 math.ceil(3.1) == 3
        # 無條件捨去 math.floor(2.9) == 2
        # C n 取 k math.comb(n, k)
        # math.gcd
        # math.lcm
        # e 次 x 冪 math.exp(x)
except EOFError:
    pass
```