Programming Contest Quick Reference

author: vanilla (unfinished)

Introduction

This document is a quick reference that gathers implementation methods for algorithms, Data Structures, Design Paradigms, Mathematics and Language specification. The source code described in this document is C++ only.

Contents

1 Disjoint-Set Data Structure

1.1 Union-Find Forest

```
int par[MAX_N];
int rnk[MAX_N];

void init(int n){
    for(int i = 0; i < n; ++i){
        par[i] = i;
        rnk[i] = 0;
    }
}

int find(int x){
    if(par[x] == x) return x;
    return par[x] = find(par[x]);
}

bool same(int x, int y){
    return find(x) == find(y);
}

void unite(int x, int y){
    if(find(x) == find(y)) return;
    if(rnk[x] < rnk[y]){
        par[x] = y;
    }else{
        par[y] = x;
        if(rnk[x] == rnk[y]) ++rnk[x];
    }
}
```

2 Shortest Path Problem

2.1 Bellman-Ford

```
#define INF (1e9)

struct Edge {
    int from, to, cost;
};

int V, E, S; // Vertex, Edge, Start Vertex
int d[MAX_V];  // Vertex Distance
Edge es[MAX_E]; // Edge Information(From, To, Cost)

void bellmanford(int S){
    fill(d, d + V, INF);
    d[S] = 0;
    while(1){
        bool update = false;
        for(int i = 0; i < E; ++i){
            Edge e = es[i];
            if(d[e.from] != INF && d[e.from] + e.cost < d[e.to]){
                d[e.to] = d[e.from] + e.cost;
                update = true;
            }
        }
        if(!update) break;
    }
}
```

2.2 Bellman-Ford(Negative cycle detection)

```
#define INF (1e9)

struct Edge {
    int from, to, cost;
};

int V, E, S; // Vertex, Edge, Start Vertex
int d[MAX_V];  // Vertex Distance
Edge es[MAX_E]; // Edge Information(From, To, Cost)

bool bellmanford_isnegative(int S){
    fill(d, d + V, INF);
    d[S] = 0;
    int count = 0;
    while(1){
        count++;
        bool update = false;
        for(int i = 0; i < E; ++i){
            Edge e = es[i];
            if(d[e.from] != INF && d[e.from] + e.cost < d[e.to]){
                d[e.to] = d[e.from] + e.cost;
                update = true;
            }
        }
        if(!update) break;
        if(count == V - 1) return true;
    }
    return false;
}
```

2.3 Dijkstra's Algorithm

```
#define INF (1e9)
const int MAX_V = 1024;
typedef pair<int, int> Edge;
vector<Edge> G[MAX_V];
int V, E;
int d[MAX_V];

void dijkstra(int s) {
    fill(d, d + MAX_V, INF);
    d[s] = 0;
    priority_queue<Edge, vector<Edge>, greater<Edge> > pq;
    pq.push(Edge(0, s));
    while(!pq.empty()) {
        int cost = pq.top().first;
        int v = pq.top().second;
        pq.pop();
        if(cost > d[v]) continue;
        for(int i = 0; i < (int)G[v].size(); ++i) {
            Edge e = G[v][i];
            if(d[e.first] > d[v] + e.second) {
                d[e.first] = d[v] + e.second;
                pq.push(Edge(d[e.first], e.first));
            }
        }
    }
}

int main() {
    cin >> V >> E;
    for (int i = 0; i < E; ++i) {
        int a, b, cost; cin>> a >> b >> cost;
        G[a].push_back(Edge(b, cost));
        G[b].push_back(Edge(a, cost));
    }
    dijkstra(0);
    for(int i = 0; i < V; ++i) cout << d[i] << " ";
    cout << endl;
}
```

3 Search Algorithm

3.1 Linear Search (Return existence)

```
int N;
int a[MAX_N];

bool linearsearch_return_existence(int target){
    for(int i = 0; i < N; ++i){
       if(a[i] == target) return true;
    }
    return false;
}
```

3.2 Linear Search (Return index)

```
int N;
int a[MAX_N];

int linearsearch_return_index(int target){
    for(int i = 0; i < N; ++i){
       if(a[i] == target) return i;
    }
    return i;
}
```

3.3 Linear Search (Standard Library)

```
find(a, a + N, target);      // Returns target Iterator
find(a, a + N, target) - a;  // Returns target Index
```

3.4 Binary Search (Return existence)(Included sort)

```
int N;
int a[MAX_N];

bool binarysearch_return_existence(int target){
    sort(a, a + N);
    int left = 0, right = N;
    while(left < right){
        int mid = left + (right - left) / 2;
        if(a[mid] == target) return true;
        else if(target < a[mid]) right = mid;
        else if(a[mid] < target) left = mid + 1;
    }
    return false;
}
```

3.5 Binary Search (Return index)(Included sort)

```
int N;
int a[MAX_N];

int binarysearch_return_index(int a[], int target){
    sort(a, a + N);
    int left = 0, right = N;
    while(left < right){
        int mid = left + (right - left) / 2;
        if(a[mid] == target) return mid;
        else if(target < a[mid]) right = mid;
        else if(a[mid] < target) left = mid + 1;
    }
    return -1;
}
```

3.6 Binary Search (Standard Library)

```
binary_search(a.begin(), a.end(), target); // Returns whether target exists in array a
```

4 Sorting Algorithm

4.1 Bubble Sort

```
void bubblesort(int arr[], int n){
   bool swapped;
   for(int i = 0; i < n - 1; i++){
     swapped = false;
     for(int j = 0; j < n - i - 1; j++){
        if(arr[j] > arr[j + 1]){
           swap(arr[j], arr[j + 1]);
           swapped = true;
        }
     }
     if (!swapped) break;
   }
}
```

4.2 Selection Sort

```
void selectionsort(int arr[], int n){
  int min, temp;
  for(int i = 0; i < n - 1; i++){
    min = i;
    for(int j = i + 1; j < n; j++){
      if(arr[j] < arr[min]){
        min = j;
      }
    }
    temp = arr[i];
    arr[i] = arr[min];
    arr[min] = temp;
  }
}
```

4.3 Quick sort(Standard Library)

```
sort(a.begin(), a.end()); //Sort array a
sort(a.begin(), a.end(), greater<int>());        // Sort by large number
```

4.4 Merge Sort

```
void merge(int arr[], int left, int mid, int right){
    int i, j, k;
    int n1 = mid - left + 1, n2 = right - mid;
    int L[n1], R[n2];

    for(i = 0; i < n1; ++i) L[i] = arr[left + i];
    for(j = 0; j < n2; ++j) R[j] = arr[mid + 1 + j];

    i = 0; j = 0; k = left;
    while(i < n1 && j < n2){
        if (L[i] <= R[j]){
            arr[k] = L[i];
            i++;
        }else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while(i < n1){
        arr[k] = L[i];
        i++; k++;
    }
    while(j < n2){
        arr[k] = R[j];
        j++; k++;
    }
}

void mergesort(int arr[], int left, int right){
    if(left < right){
        int mid = left + (right - left) / 2;
        mergesort(arr, left, mid);
        mergesort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

5 Mathematics

5.1 Greatest Common Divisor(Standard Library)

```
__gcd(x, y);                    //returns gcd of x and y.
```

5.2 Greatest Common Divisor(Euclidean algorithm)

```
int gcd(int x, int y) {
    return !y ? x : gcd(y, x % y);
}
```

5.3 Least Common Multiple

```
int lcm(x, y){
    return (x*y)/__gcm(x, y);
}
```

5.4 Divisors

```
vector<int> divisors(int n) {
    vector<int> res;
    for(int i = 1; i*i <= n; ++i) {
        if(n % i != 0) continue;
        res.push_back(i);
        if(n/i == i) continue;
        res.push_back(n/i);
    }
    return res;
}
```

5.5 Primality test(Simple)

```
bool isprimenumber_pseudocode(int x){
  if(x < 2) return false;
  else if(x == 2) return true;
  if(x % 2 == 0) return false;
  for(int i = 3; i <= x / i; i += 2){
    if(x % i == 0) return false;
  }
  return true;
}
```

5.6 Primality test(Sieve of Eratosthenes)

```
bool isprimenumber_sieve_of_eratosthenes(int x){
  bool isPrime[x+1];
  fill(isPrime, isPrime + x + 1, false);
  for(int i = 3; i < x + 1; i += 2) isPrime[i] = true;
  isPrime[2] = true;
  for(int i = 3; i * i < x + 1; i += 2){
    if(isPrime[i]){
      for(int j = i * i; j < x + 1; j += i){
        isPrime[j] = false;
      }
    }
  }
  return isPrime[x];
}
```

5.7 Prime factorization(Simple)

```
vector<int> primefactorization_simple(int x){
  vector<int> primes;
  int k = 0;
  int l = ceil(sqrt(x));
  if(x > 3){
    while(x != 1 && x % 2 == 0){
        primes.push_back(2);
        x /= 2;
        k++;
      }
      int i = 3;
      while(x != 1 && i <= l){
        while(x != 1 && x % i == 0){
          primes.push_back(i);
          x /= i;
          k++;
        }
        i += 2;
      }
      if(k >= 1 && x > 1){
        k++;
        primes.push_back(x);
      }
  }
  if(primes.empty()) primes.push_back(x);
  return primes;
}
```

6 Language specification

6.1 Count target in array(Returns int)

```
count(a.begin(), a.end(), Target);   // Returns Number of Target
```

6.2 Whether two arrays are equivalent(Returns bool)

```
equal(a.begin(), a.end(), b.begin());   // Returns whether two arrays are equivalent
```

6.3 Search substring(Returns iterator)

```
search(a.begin(), a.end(), b.begin(), b.end());       // Searching ArrayB in ArrayA
```

6.4 Search substring(Returns index)

```
a.find("string");         // Searching "string" in string a
```

6.5 Swap two variables

```
swap(a, b);         // Swapping a and b
```

6.6 Replace elements

```
replace(a.begin(), a.end(), 2, 4);               // a = {1, 2, 3, 2, 1} -> {1, 4, 3, 4, 1}
```

6.7 Remove specific elements

```
w.erase(remove(a.begin(), a.end(), 2), a.end());         // a = {1, 2, 3, 2, 1} -> {1, 3, 1}
```

6.8 Remove duplicate elements

```
// Require sort
v.erase(unique(a.begin(), a.end()), a.end());         // a = {1, 1, 2, 2, 3, 3} -> {1, 2, 3}
```

6.9 Rotate elements

```
rotate(a.begin(), a.begin() + 2, a.end());       // a = {1, 2, 3, 4, 5} -> {3, 4, 5, 1, 2}
```

6.10 Shuffle elements

```
random_shuffle(a.begin(), a.end());  //Random shuffling
```

6.11 Lower bound, Upper bound

```
// v = { 1, 2, 2, 3, 3 }
lower_bound(v.begin(), v.end(), 2); // v.begin() + 1
upper_bound(v.begin(), v.end(), 2); // v.begin() + 3
```

6.12 Min, Max

```
min(1, 2);   // Returns small value
max(3, 4);   // Returns large value
```

6.13 Min, Max elements

```
// v = { 3, 5, 2, 4, 1 }
max_element(v.begin(), v.end()); // v.begin() + 1
*max_element(v.begin(), v.end()); // 5
max_element(v.begin(), v.end()) - v.begin(); // 1
```

6.14 Permutation

```
do{
  // Array state
}while(next_permutation(a.begin(), a.end()));

do{
  // Array state
}while(prev_permutation(a.begin(), a.end()));
```

6.15 Accumulate

```
// a = { 1, 2, 3, 4, 5 }
// b = { "He", "llo", "Wor", "ld" }
accumulate(a.begin(), a.end(), 0); // 15 (int)
accumulate(a.begin(), a.end(), 0ll); // 15 (long long)
accumulate(a.begin(), a.end(), 1, multiplies<int>()); // 120
accumulate(b.begin(), b.end(), string()); // "HelloWorld"
```

6.16 Whether empty

```
a.empty();   // Returns whether array a is empty
```

6.17 Initialization matrix

```
fill((int *)G, (int *)G + (MAX_H * MAX_W), INF);
```

7 Standard Template Library(STL)

7.1 Correspondence table of data structures and methods

Data structure	Insert front	Return front	Pop front	Insert back	Return back	Remove back	Insert
list	push_front()	front()	pop_front()	push_back()	back()	pop_back()	
deque	push_front()	front()	pop_front()	push_back()	back()	pop_back()	
vector		front()		push_back()	back()	pop_back()	
queue		front()	pop()	push()	back()		
stack	push()	top()	pop()				
priority_queue		top()	pop()				push()