Dijkstra Código

```
vi dist(V, INF); dist[s] = 0;                    // INF = 1B to avoid overflow
priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));
                          // ^to sort the pairs by increasing distance from s
while (!pq.empty()) {                                          // main loop
  ii front = pq.top(); pq.pop();     // greedy: pick shortest unvisited vertex
  int d = front.first, u = front.second;
  if (d == dist[u])            // this check is important, see the explanation
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                       // all outgoing edges from u
      if (dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second;                // relax operation
        pq.push(ii(dist[v.first], v.first));
}   } }  // note: this variant can cause duplicate items in the priority queue
```

Bellman-Ford Código

```
vi dist(V, INF); dist[s] = 0;
for (int i = 0; i < V - 1; i++)  // relax all E edges V-1 times, overall O(VE)
  for (int u = 0; u < V; u++)                          // these two loops = O(E)
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];        // we can record SP spanning here if needed
      dist[v.first] = min(dist[v.first], dist[u] + v.second);          // relax
    }
```

Example codes: ch4_06_bellman_ford.cpp; ch4_06_bellman_ford.java

```
bool hasNegativeCycle = false;
for (int u = 0; u < V; u++)                              // one more pass to check
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dist[v.first] > dist[u] + v.second)         // if this is still possible
      hasNegativeCycle = true;                     // then negative cycle exists!
  }
printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes" : "No");
```