

1

R3: $<, >, <=, >=$

ordre lexicographie sur les pair est déjà dans la stl

1.1 Multimaps

(Requires a less-than comparison function.)

$O(3n_T)$: jede Kante jedes Dreiecks als Schlüssel

$|K| \cdot O(\log(n_T))$: Zugriff auf Container für jede Kante

$\Rightarrow O(n_T \log(n_T))$

speichern in 2dim Array: Zugriff in konstanter Zeit

unordered map(hash tables) has constant time performance on all operations provided no collisions occur. When collisions occur, traversal of a linked list containing all elements of the same bucket (those that hash to the same value) is necessary, and in the worst case, there is only one bucket; hence $O(n)$

1.2 Listen

$O(3n_T)$, push front(): jede Kante jedes Dreiecks einfügen

$O(N \log N)$, container size N : sort(): definiere hierfür $<$ für R3 (Nach ersten und dann nach zweitem Element sortieren etc)

doppelt verlinkt! und dann mit Kante und Sommet prev und next von Listenelement!

konstanter Zugriff mit Iterator!

$O(3n_T)$: pop front und speichern in 2 dim array

Alternative für 2D array für konstanten Zugriff?

Daten könnten direkt in 2D array gespeichert werden, Listen und Maps überflüssig

1.3 Fläche

orientierter Flächeninhalt

$$(a_i, b_i, p) = (a_i \times b_i) \cdot p = \det(a_i, b_i, p)$$

bei drei Vektoren in Zeilen