Introduction of rucgraph

Folder: build_in_progress

This folder contains informal codes.

Folder: data_structures

This folder contains some special data structures.

PairingHeapYS.h

This file contains a pairing heap.

PairingHeapYS_with_offset.h

This file contains an augmented pairing heap. In this heap, there is an offset value for every inside node. Using these values, we can change the key values of all inside nodes in O(1) time!

Union_Find.h

This file contains the Union Find data structure.

Folder: dgraph_v_of_v

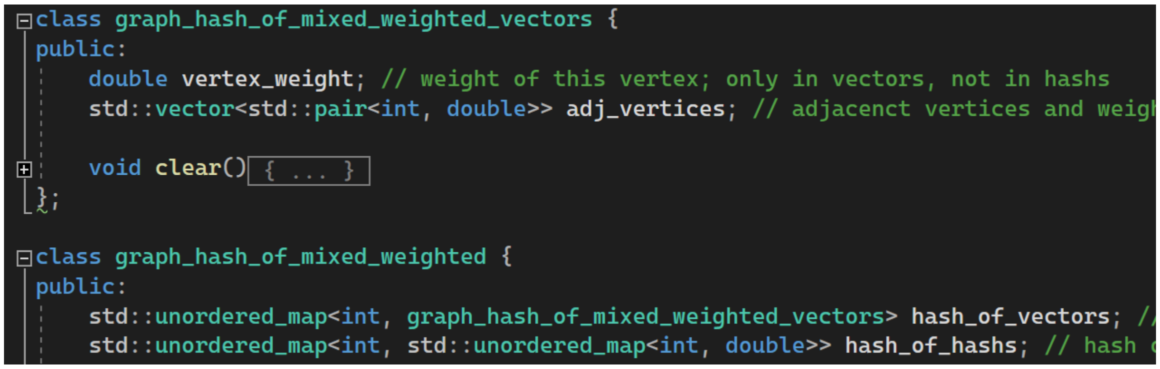
dgraph_v_of_v.h

This is an adjacency list built using vectors, for representing a directed graph with edge weights.

Folder: graph_hash_of_mixed_weighted

graph_hash_of_mixed_weighted.h

This is a complex adjacency list. Its structure is kind of between hash of vectors and hash of hashes. If a vertex has a large degree, then its adjacency edges and edges weights are stored in a sorted vector, otherwise stored in a hash. This complex nature is to balance the space and search time for achieving an ideal time complexity.



```
class graph_hash_of_mixed_weighted_vectors {
public:
    double vertex_weight; // weight of this vertex; only in vectors, not in hashs
    std::vector<std::pair<int, double>> adj_vertices; // adjacenct vertices and weig

    void clear() { ... }
};

class graph_hash_of_mixed_weighted {
public:
    std::unordered_map<int, graph_hash_of_mixed_weighted_vectors> hash_of_vectors; //
    std::unordered_map<int, std::unordered_map<int, double>> hash_of_hashs; // hash
```

graph_hash_of_mixed_weighted_binary_operations.h

This is to binary operate a sorted vector of edges, for achieving a high efficiency.

graph_hash_of_mixed_weighted_update_vertexIDs.h
This is change vertex IDs in a graph built using graph_hash_of_mixed_weighted.

Folder: common_algorithms
graph_hash_of_mixed_weighted_connected_components.h
This is to find the maximum connected components of a graph. It returns the maximum connected components with the structure of list of lists (or vector of vectors) of vertices.

graph_hash_of_mixed_weighted_minimum_spanning_tree.h
This is to find a minimum spanning tree of a graph.

graph_hash_of_mixed_weighted_random_spanning_tree.h
This is to find a random spanning tree of a graph.

graph_hash_of_mixed_weighted_shortest_paths.h
This is to find shortest paths.

Folder: extract_subgraph
graph_hash_of_mixed_weighted_breadth_first_search_a_fixed_depth_of_vertices.h
This is to breadth_first_search a_fixed_depth_of_vertices, usually used in experiments.

graph_hash_of_mixed_weighted_breadth_first_search_a_fixed_number_of_vertices.h
This is to breadth_first_search a_fixed number of_vertices.

Note that, we assume that a fixed number of vertices can be searched, otherwise it returns as many vertices as possible.

graph_hash_of_mixed_weighted_breadth_first_search_a_fixed_number_of_vertices_in_unconnected_graphs.h
This is to breadth_first_search a_fixed number of_vertices in disconnected graphs. It may randomly search multiple components.

graph_hash_of_mixed_weighted_breadth_first_search_a_fixed_number_of_vertices_in_unconnected_graphs_start_from_maxcpn.h
This is to breadth_first_search a_fixed number of_vertices in disconnected graphs. It search components in a decreasing order of the sizes of components. This is frequently used in experiments.

graph_hash_of_mixed_weighted_breadth_first_search_a_set_of_vertices.h
This is to breadth_first_search a set of_vertices.

graph_hash_of_mixed_weighted_breadth_first_search_a_tree.h
This is to breadth_first_search a tree.

graph_hash_of_mixed_weighted_breadth_first_search_a_tree_of_edges.h
This is to breadth_first_search a tree of edges.

graph_hash_of_mixed_weighted_extract_subgraph.h

This is to extract a subgraph for a given set of vertices.

graph_hash_of_mixed_weighted_random_walk_a_fixed_number_of_vertices_in_unconnected_graphs_start_from_maxcpn.h

This is to random walk a_fixed number of_vertices in disconnected graphs. It search components in a decreasing order of the sizes of components. This is frequently used in experiments.

Folder: random_graph

graph_hash_of_mixed_weighted_generate_random_connected_graph.h

This is to generate a random connected graph.

graph_hash_of_mixed_weighted_generate_random_graph.h

This is to generate a random (may not be connected) graph.

graph_hash_of_mixed_weighted_generate_random_graph_for_Cytoscape.cpp

This is an example cpp file for generate and save a random graph that can be draw on Cytoscape.

Folder: read_save

graph_hash_of_mixed_weighted_binary_save_read.h

This is to save and read a graph_hash_of_mixed_weighted in binary format.

graph_hash_of_mixed_weighted_read_for_GSTP.h

This is to read two graphs and a group of vertex IDs and a double value, for testing codes of solving group Steiner trees.

graph_hash_of_mixed_weighted_read_graph_with_weight.h

This is to read a graph that is saved in readable format.

graph_hash_of_mixed_weighted_save_for_GSTP.h

This is to save two graphs and a group of vertex IDs and a double value, for testing codes of solving group Steiner trees.

graph_hash_of_mixed_weighted_save_graph_with_weight.h

This is to save a graph in readable format.

Folder: two_graphs_operations

graph_hash_of_mixed_weighted_copy_graph_to_another_graph.h

This is to merge a graph into another graph, both in the format of graph_hash_of_mixed_weighted.

graph_hash_of_mixed_weighted_copy_weights_of_graph1_to_graph2.h

This is to copy vertex and edge weights in a graph into another graph, both in the format of graph_hash_of_mixed_weighted.

graph_hash_of_mixed_weighted_graph1_is_graph2.h
This is to check whether two graphs are the same.

graph_hash_of_mixed_weighted_graph1_is_in_graph2.h
This is to check whether the vertices and edges in a graph are in another graph or not. It does not check weights.

graph_hash_of_mixed_weighted_merge_graph_hash_of_mixed_weighted.h
This is to merge a graph into another graph, both in the format of graph_hash_of_mixed_weighted.

graph_hash_of_mixed_weighted_to_graph_v_of_v_idealID.h
This is to change a graph_hash_of_mixed_weighted to a graph_v_of_v_idealID. This function inputs a vertex mapping function.

graph_hash_of_mixed_weighted_to_graph_v_of_v_idealID_2.h
This is to change a graph_hash_of_mixed_weighted to a graph_v_of_v_idealID. This function does not input a vertex mapping function, but inputs a max_ID value.

Folder: weight_operations

graph_hash_of_mixed_weighted_ec_update_pairwise_jaccard_distance.h
This is to update edge weights to pairwise_jaccard_distance.

graph_hash_of_mixed_weighted_nw_ec_normalization.h
This is to normalize vertex and edge weights.

graph_hash_of_mixed_weighted_sum_of_nw_ec.h
This is to compute the sum of vertex and/or edge weights.

Folder: graph_hash_of_vectors_unweighted

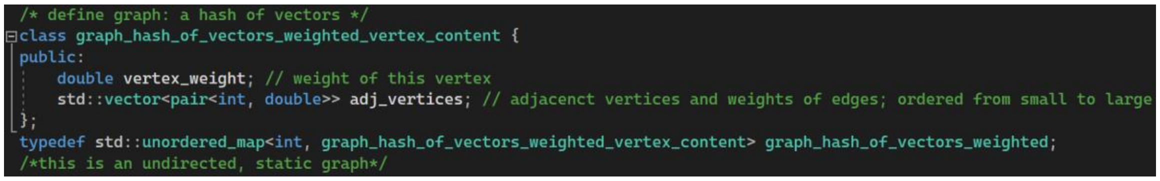
graph_hash_of_vectors_unweighted.h
This is an adjacency list build using a hash of vectors:

```
/* define graph: a hash of vectors */
typedef std::unordered_map<int, std::vector<int>> graph_hash_of_vectors_unweighted;
```

This adjacency list does not contain vertex or edge weights.

Folder: graph_hash_of_vectors_weighted

graph_hash_of_vectors_weighted.h
This is an adjacency list build using a hash of vectors:



```
/* define graph: a hash of vectors */
class graph_hash_of_vectors_weighted_vertex_content {
public:
    double vertex_weight; // weight of this vertex
    std::vector<pair<int, double>> adj_vertices; // adjacenct vertices and weights of edges; ordered from small to large
};
typedef std::unordered_map<int, graph_hash_of_vectors_weighted_vertex_content> graph_hash_of_vectors_weighted;
/*this is an undirected, static graph*/
```

This adjacency list contains vertex or edge weights.

Folder: graph_v_of_v_idealID

graph_v_of_v_idealID.h

This is an adjacency list with the structure of vectors of vectors.

graph_v_of_v_idealID_change_new_vertexIDs.h

This is to change vertex IDs in a graph built using graph_v_of_v_idealID.

Folder: common_algorithms

graph_v_of_v_idealID_connected_components.h

This to find the maximum connected components of a graph built using graph_v_of_v_idealID. It returns the maximum connected components with the structure of list of lists of vertices.

graph_v_of_v_idealID_shortest_paths.h

This is to find shortest paths in a graph built using graph_v_of_v_idealID.

Folder: extract_subgraph

graph_v_of_v_idealID_breadth_first_search_a_set_of_vertices.h

This is to breadth_first_search a_set_of_vertices from a root vertex in a graph built using graph_v_of_v_idealID.

graph_v_of_v_idealID_extract_subgraph.h

This is to extract a subgraph based on a given set of vertices.

Folder: random_graph

graph_v_of_v_idealID_generate_random_connected_graph.h

This is to generate a random connected graph built using graph_v_of_v_idealID.

Folder: read_save

graph_v_of_v_idealID_read_for_GSTP.h

This is to read two graphs and a group of vertex IDs, for testing codes of solving group Steiner trees. The read files are generated by the following codes.

graph_v_of_v_idealID_save_for_GSTP.h

This is to save two graphs and a group of vertex IDs, for testing codes of solving group Steiner trees.

Folder: text_mining

binary_save_read_vector

This is to save and read vectors in binary format. Notably, the elements in vectors should have fixed sizes.

binary_save_read_vector_of_vectors.h

This is similar to the above file, for saving and reading vectors of vectors.

convert_number_to_array_of_binary.h

This is to concert a number to an array of binary values, e.g., from 3 to 11.

latitude_and_longitude_distance.h

This is to compute the distance between two points using latitude_and_longitude.

list_all_files_in_a_directory.h

This is to list all file names in a path.

parse_string.h

This is to parse a string based on a delimiter.

parse_substring_between_pairs_of_delimiters.h

This is get substrings between a pair of different delimiters.

parse_substring_between_two_unique_delimiters.h

This is to get the substring between two_unique_delimiters.

print_items.h

This is used to print items.

read_csv.h

This is to read a csv file into a vector of vectors of strings.

read_file_line_by_line.h

This is to print a file line by line.

read_file_total_line_number.h

This is to print the total line number of a file.

replace_chars_in_string.h

This function replace all chars "from" in a string to "to".

string_contains_number.h

This is to check whether a string contains a number char.

string_is_number.h

This is to check whether a string is a number.

StringCompare_caseInSensitive.h

This is an insensitive comparison of two strings, e.g., A==a.

utc_time_to_local_time.h

This is to convert utc_time_to_local_time.

Folder: tool_functions

This folder contains some tool functions.

Combinations_Permutations.h

This file contains codes to enumerate every possible permutation of a set of elements.

Current_Memory_Consumption_of_This_Process.h

This file is to check how many RAM has been allocated by the OS to the current process.

ThreadPool.h

This is a widely adopted ThreadPool.h implementation.

ThreadPool2.h

ThreadPool uses mutex, while ThreadPool2 uses shared_mutex.

ThreadPool is faster than ThreadPool2 on Linux, but is slower than ThreadPool2 on Windows.