Greedy Method

Greedy Matching
Minimum Spanning Tree

Knapsack
Dijkstra's Single-Source Shortest-Path

Manisha
Asst. Prof.comp sc.

Greedy Method

The Greedy Concept

Makes the choice that looks best at the moment.

Hopes that "local optimal" choices lead to "global optimal'. solution.

Two basic properties of optimal greedy algorithms

Optimal Substructure Property: A problem has optimal substructure if an optimal solution to the problem contains within it optimal solutions to its sub problems.

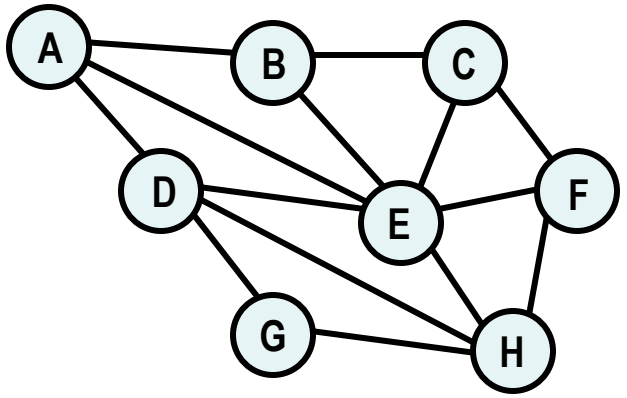
Greedy Choice Property: If a local greedy choice is made, then an optimal solution including this choice is possible.

Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = { }
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end



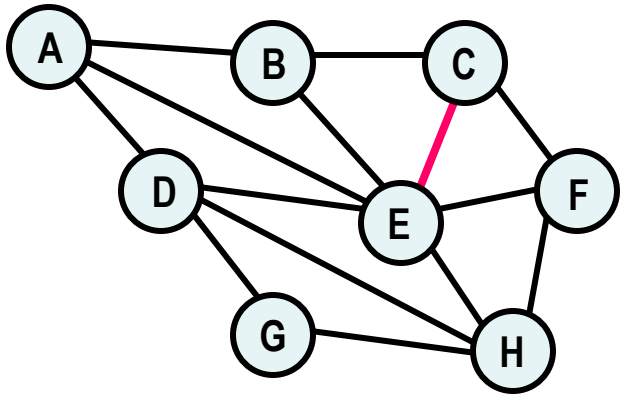
The Annals of Applied Probability, Vol. 3, No. 2. (May, 1993), pp. 526-552

Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end

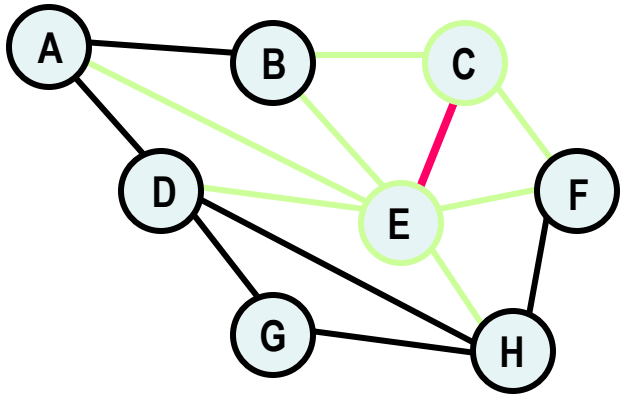


Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end

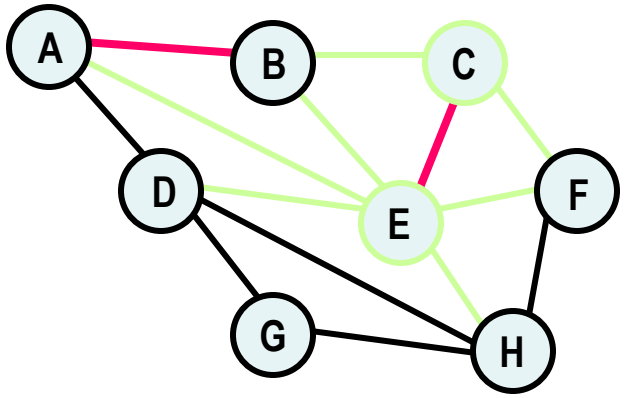


Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE),(AB)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end

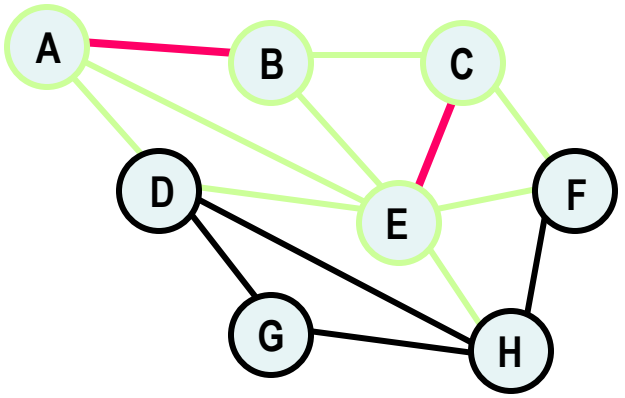


Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE),(AB)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end

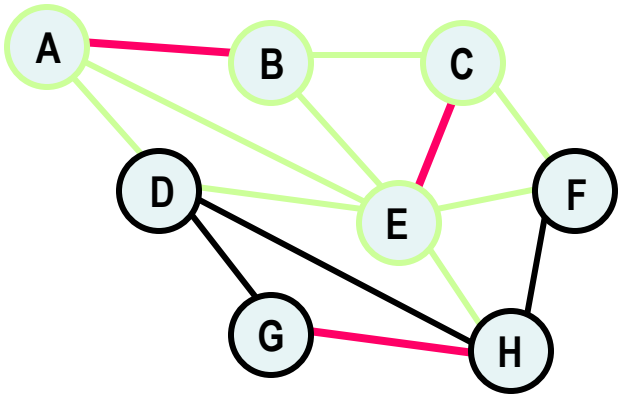


Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE),(AB),(GH)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end

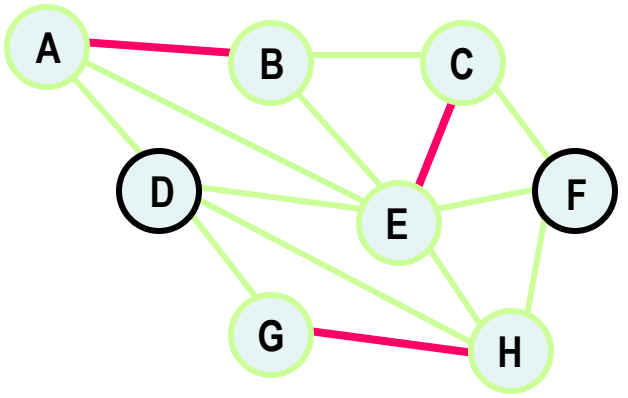


Greedy Matching

In this paper we discuss the expected performance of the simplest of matching algorithms, that is, the GREEDY (or myopic) algorithm. Given a graph G, the algorithm repeatedly chooses an edge e and deletes it along with its end vertices until the graph remaining has no edges. The set of chosen edges forms a matching. More formally the algorithm proceeds as follows:

M = {(CE),(AB),(GH)}
V = { A B C D E F G H }
E = {(AB),(AD),(AE),(BC),(BE),(CE),(CF),
(DE),(DG),(DH),(EF),(EH),(FH),(GH)}

begin
  $M \leftarrow \emptyset$;
  while $E(G) \neq \emptyset$ **do**
  begin
    A: Choose $e = \{u, v\} \in E$
      $G \leftarrow G \setminus \{u, v\}$;
      $M \leftarrow M \cup \{e\}$
  end;
Output M
end



When Does the Greedy Method Work?

When using the U.S. minted coin set, we find that the coin-changing algorithm gives us the minimum number of coins. However, this greedy algorithm does not work when we replace the quarter with a 26-cent piece.

See if you can determine the necessary relationships between coin values in a coin set for which the greedy coin-changing algorithm will result in a minimum number of coins for any initial_value.

For your rules to be correct they must hold up under the following evaluations:

1. See if you can find a coin set that follows the rules but doesn't support the greedy algorithm.

2. Alternately you can attempt to find a coin set that supports the greedy algorithm but is not represented by the rules.

Prim's Algorithm

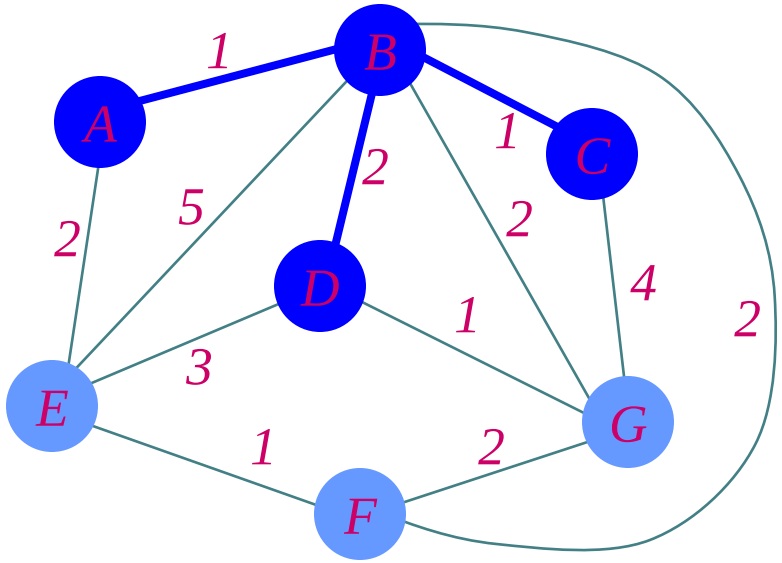
Given a weighted graph G consisting of a set of vertices V and a set of edges E with weights, where $e = (v_i, v_j) \in E$ *and* $v_i, v_j \in V$ *Prepare a vertex set and an edge set to hold elements selected by Prim's Algorithm.*

1. Choose an arbitrary starting vertex v_j

2. Find the smallest edge e incident with with a vertex in the vertex set whose inclusion in the edge set does not create a cycle.

3. Include this edge in the edge list and its vertices in the vertex list.

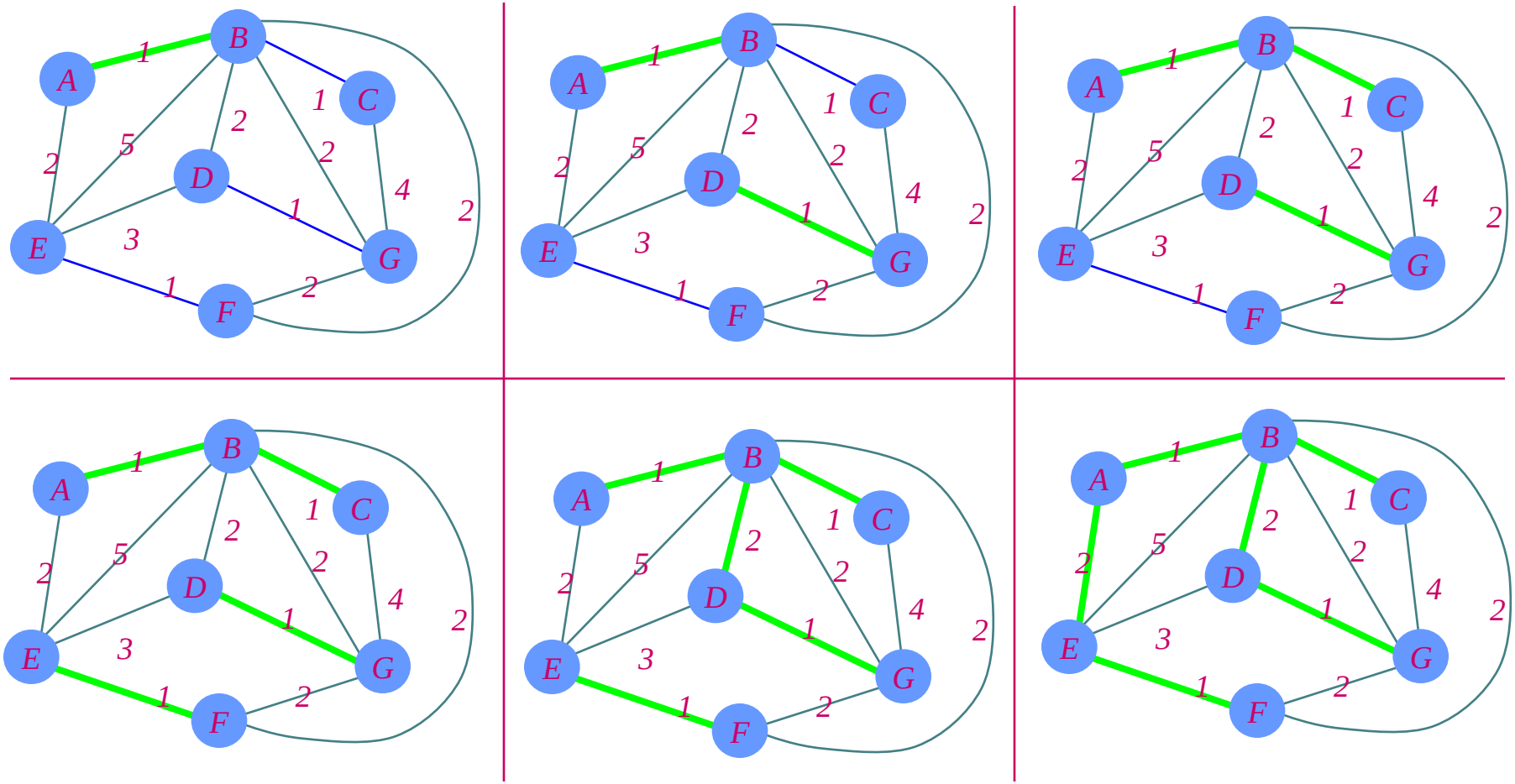
4. Repeat Steps 2 and 3 until all vertices are in the vertex list.



Exercise Prim's Algorithm on this example weighted graph starting with vertex B and again starting at vertex F. Did you get the same spanning tree? If the trees were distinct, did they have the same value?

Kruskal's Algorithm

The minimum spanning tree problem can also be solved using Kruskal's Algorithm. In this approach, we simply choose minimum-weight edges from the graph so long as an edge does not create a cycle in the edge set. We stop choosing edges when every vertex is a node for at least one of the edges in the set and the tree is connected.



Knapsack Problem

A thief can carry a maximum weight in his knapsack and he wants to maximize the amount of value he carries. There is a given amount of each type of item and each item has a specified total value. How much of each item should he carry to get the maximum value? Maximum weight = 75.

Item #	Total Weight	Total Value	V/W	Order Taken	Fract Taken
1	20	10	1/2	4	0.0
2	40	25	5/8	2	1.0
3	10	15	3/2	1	1.0
4	50	30	3/5	3	0.5
5	30	10	1/3	5	0.0

Total Weight	Total Value
75	55

If we compute the ratio of the total value to the total weight, we can arrange the items in the order of preference. Then we can take the most valuable items first. Eventually we will fill our knapsack by taking as much of the last item as will fit.

0/1 Knapsack Problem

We now consider the case in which fractional portions of an item are not allowed. That is the thief must take all or none of each item.

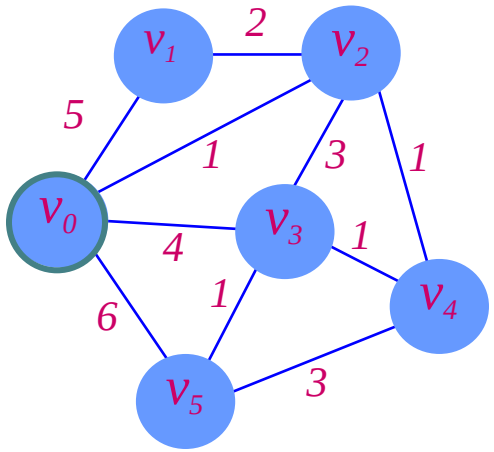
Item #	Total Weight	Total Value	V/W	Rank Order	Item Value
1	20	10	1/2	4	
2	40	25	5/8	2	25
3	10	15	3/2	1	15
4	50	30	3/5	3	
5	30	10	1/3	5	

Total Weight	Total Value
70	50

This is called the 0/1 knapsack problem. As shown in this example, the greedy method does not apply to this problem. The third most valuable item is too heavy to be added to the knapsack. Develop an example 0/1 knapsack problem in which the most valuable item is not an item in the optimal solution. (Weight of most valuable item cannot exceed weight limit by itself.)

Single Source Shortest Path

Given a weighted graph G find the minimum weight path from a specified vertex v_0 to every other vertex.



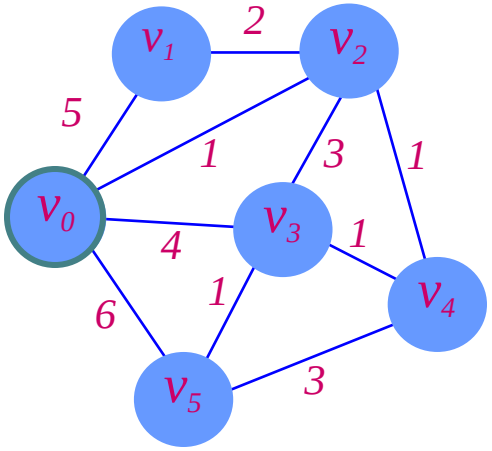
The *single source shortest path* problem is as follows. We are given a directed graph with *nonnegative* edge weights $G = (V,E)$ and a distinguished *source vertex*, $s \in V$. The problem is to determine the distance from the source vertex to every vertex in the graph.

Dijkstra's Algorithm for SSSP

```
Dijkstra(G,w,s)
{
    for (each u ∈ V)
    {
        d[u] = ∞;
        color[u] =white;
    }
    d[s] = 0;
    pred[s] = NIL;
    Q = (queue with all vertices);

    while (Non-Empty(Q))
    {
        u = Extract-Min(Q);
        for (each v ∈ Adj[u])
            if (d[u] + w(u,v) < d[v])
            {
                d[v] = d[u] + w(u,v);
                Decrease-Key(Q, v, d[v]);
                pred[v] = u;
            }
        color[u] =black;
    }
}
```

	v1	v2	v3	v4	v5
	5	**1**	4	–	6
{2}	3		**4**	**2**	6
{24}	3		3		5
{241}			**3**		**5**
{2413}					**4**



Summary

Greedy Matching

Minimum Spanning Tree

Prim's Algorithm

Kruskal's Algorithm

Knapsack

Fractional Knapsack

0/1 Knapsack

Diskstra's Single-Source Shortest Path Algorithm