アルゴリズム論
Theory of Algorithms

第4回講義
貪欲アルゴリズム

アルゴリズム論
Theory of Algorithms

Lecture #4
Greedy Algorithm

貪欲法(グリーディー法)

最適化問題を解くときに有効な設計手法.
各時点で全体的なことは考えず，最良のものを選択.

問題P11: (硬貨の交換問題)

硬貨の種類を50円玉，10円玉，5円玉，1円玉とする.
これらの硬貨で枚数が最小になるようにn円を交換するにはどのように硬貨を選べばよいか.

アルゴリズムP11-A0:

貪欲法は各時点で局所的に最善の選択をする.
まず，最も大きい硬貨である50円玉でできるだけ払い，
残りを次に大きい10円玉で払い，さらにその残りを
5円玉で払う. まだ残りがあれば，最後に1円玉で払う.

例 : n = 127の場合.

50円２枚(100円)，10円２枚(120円)，5円１枚(125円)，1円２枚

Greedy Algorithms

a group of algorithms effective for solving optimization problems.
at each iteration, make a locally best selection without considering its global effect

Problem P11: (Coin Exchange problem)

Suppose we have coins of 50 yen, 10 yen, 5 yen and 1 yen.
How should we choose a minimum number of coins to exchange n yens?

Algorithm P11-A0:

A greedy algorithm makes locally optimal selection at each iteration.
First, we pay by as many 50-yen coins, largest coins, and then pay by 10-yen coins the next largest ones, and further by 5-yen coins. If there still remains any, we pay by 1-yen coins.

Example : the case of n = 127

50-yen coins x 2(100yen), 10-yen x 2(120 yen),
5-yen x 1(125yen), 1-yen x 2

硬貨の交換問題

50円硬貨の枚数を M50,
10円硬貨の枚数を M10,
5円硬貨の枚数を M5,
1円硬貨の枚数を M1

とすれば，n円が与えられたとき，各硬貨の枚数は

M50 = n/50; n = n mod 50;
M10 = n/10; n = n mod 10;
M5 = n/5; n = n mod 5;
M1 = n;

として求まる.

そのうち
レポート課題にて

演習問題:
上記の方法で常に最小枚数の硬貨が求まることを証明せよ.

演習問題:
30円硬貨も使えるときにも上記の方法で最適解が求まるか？

Coin Exchange Problem

Let the number of 50-yen coins be M50,
Let the number of 10-yen coins be M10,
Let the number of 5-yen coins be M5,
Let the number of 1-yen coins be M1.

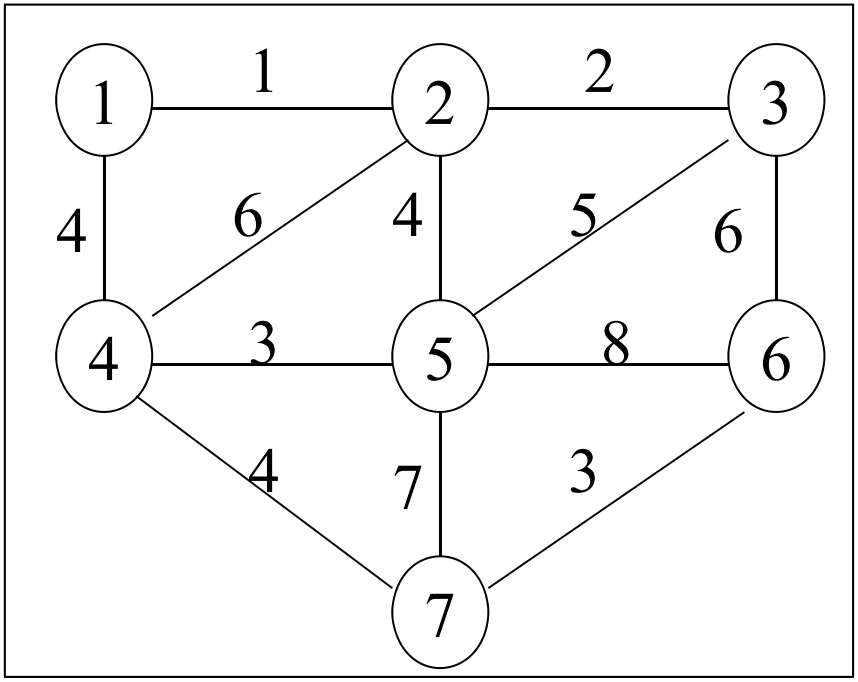
Then, give n yen, the numbers of coins are given by

M50 = n/50; n = n mod 50;
M10 = n/10; n = n mod 10;
M5 = n/5; n = n mod 5;
M1 = n;

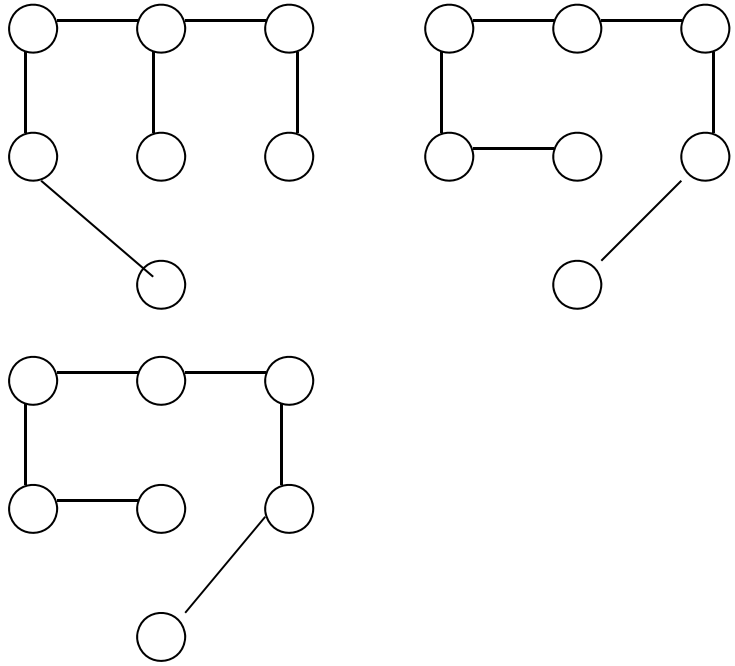
Exercise: Prove that the above algorithm always finds a smallest number of coins.

Exercise: Does the above algorithm work when 30-yen coins are available as well?

問題P12:(最小全域木問題)
辺に重みが付いた(無向)グラフが与えられたとき，全ての点を含む木(全域木)のなかで辺の重みの和が最小となるものを見つけよ.

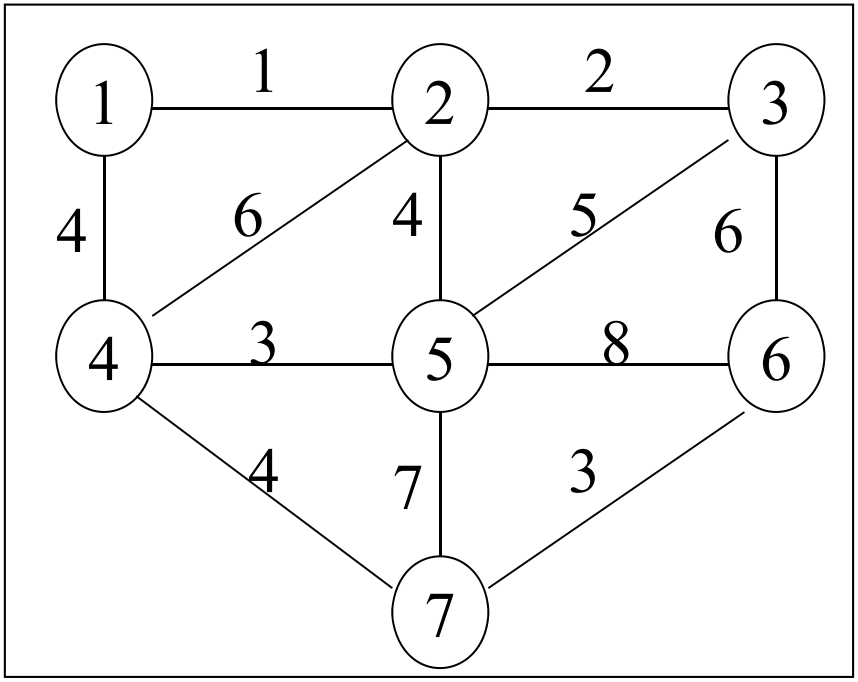


与えられた重みつきグラフ

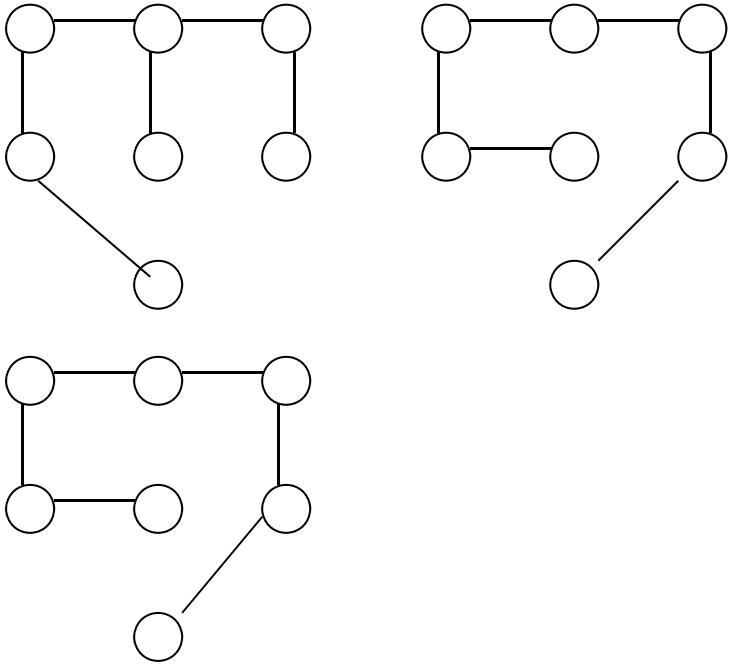


様々な全域木

Problem P12:(Minimum Spanning Tree Problem)
Given an undirected graph with weighted edges, find a tree containing all the vertices (spanning tree) with the smallest total edge weight.



given weighted graph



various spanning trees

問題P12:(最小全域木問題)
辺に重みが付いた(無向)グラフが与えられたとき，全ての点を含む木(全域木)のなかで辺の重みの和が最小となるものを見つけよ.

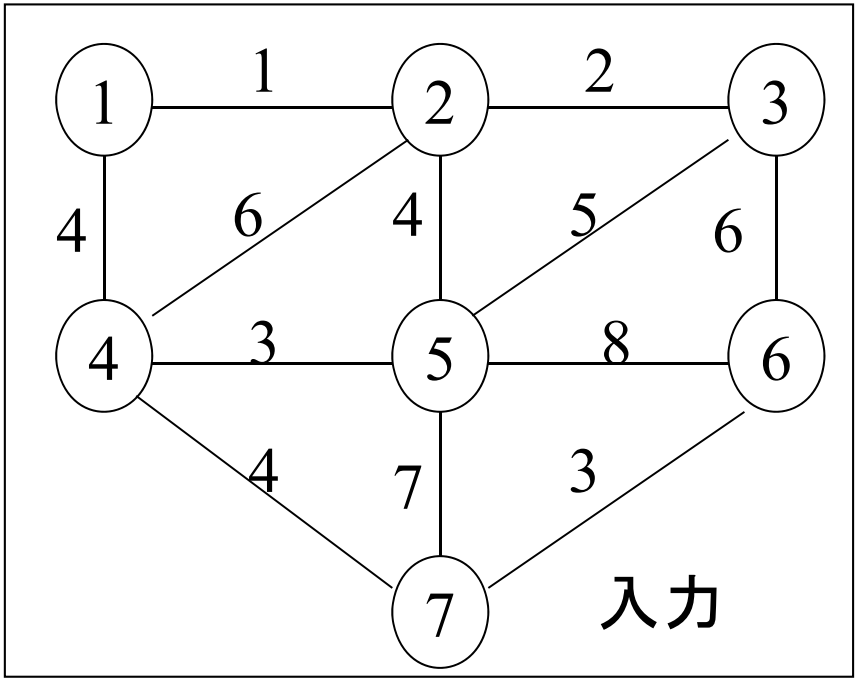
重み付き連結無向グラフ　G(V,E,c)
　V: 頂点の集合，E:(無向)辺の集合，c(e):辺eの重み>0.

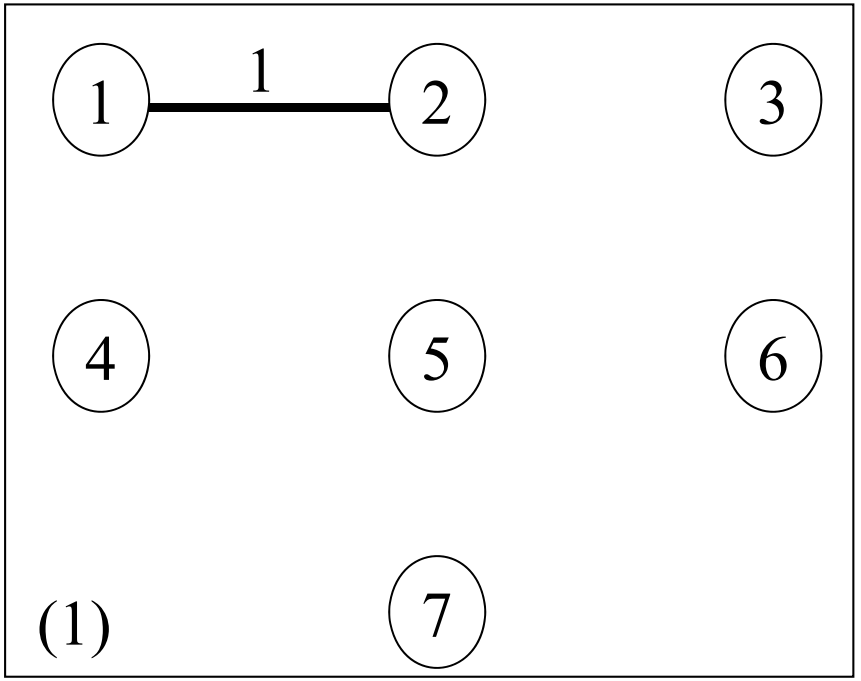
アルゴリズムP12-A0:(Kruskalのアルゴリズム)
T = 空集合;
while(Eは空でない){
　Eの中から重み最小の辺eを選び，Eからeを取り除く;
　Tに辺eを加えてできるグラフがサイクルを含まなければ，
　Tにeを加える;
}
辺集合Tによって定まるグラフ(木)を最小全域木として出力;

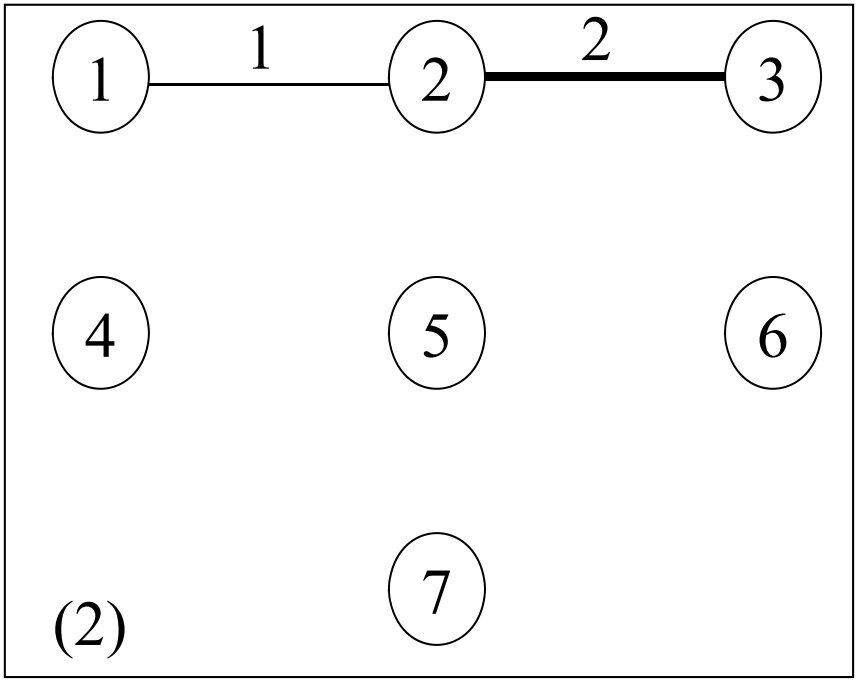
Problem P12:(MST: Minimum Spanning Tree Problem)
Given an undirected graph with weighted edges, find a tree containing all the vertices (spanning tree) with the smallest total edge weight.

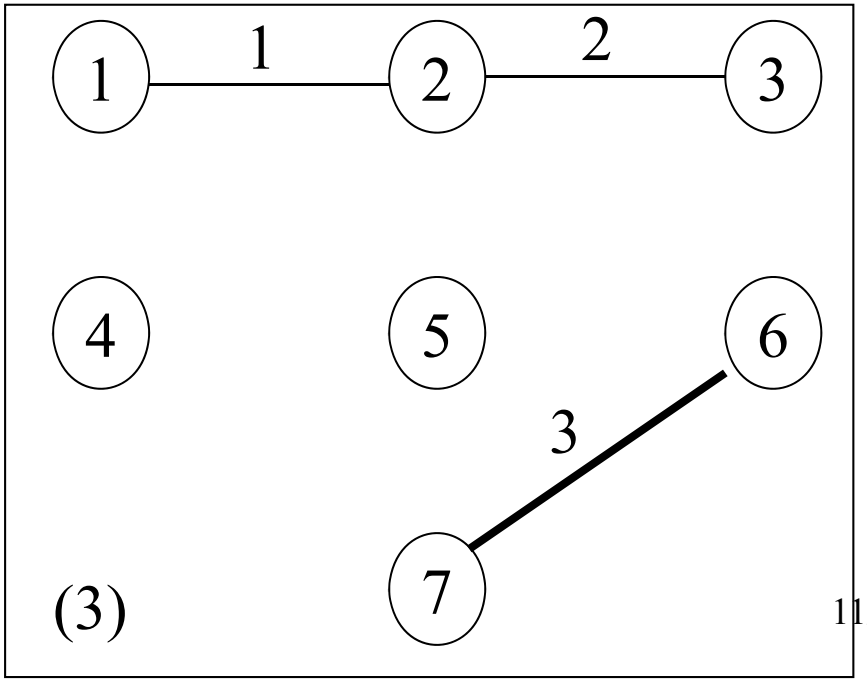
undirected connected graph with edge weights G(V,E,c)
V: set of vertices, E: set of (undirected) edges,
c(e): weight of an edge e >0.

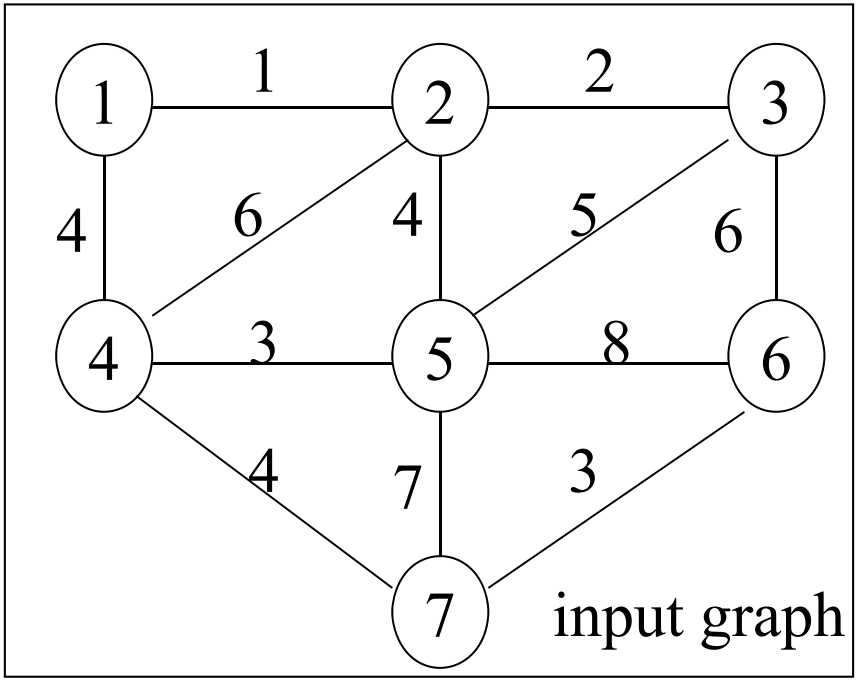
Algorithm P12-A0:(Kruskal's algorithm)
T = empty set;
while(E is not empty){
Choose an edge *e* of smallest weight, and remove it from E;
Add the edge *e* to T if the graph obtained by adding e to T does not contain any cycle;
}
Output the graph (tree) determined by the edge set T as MST;

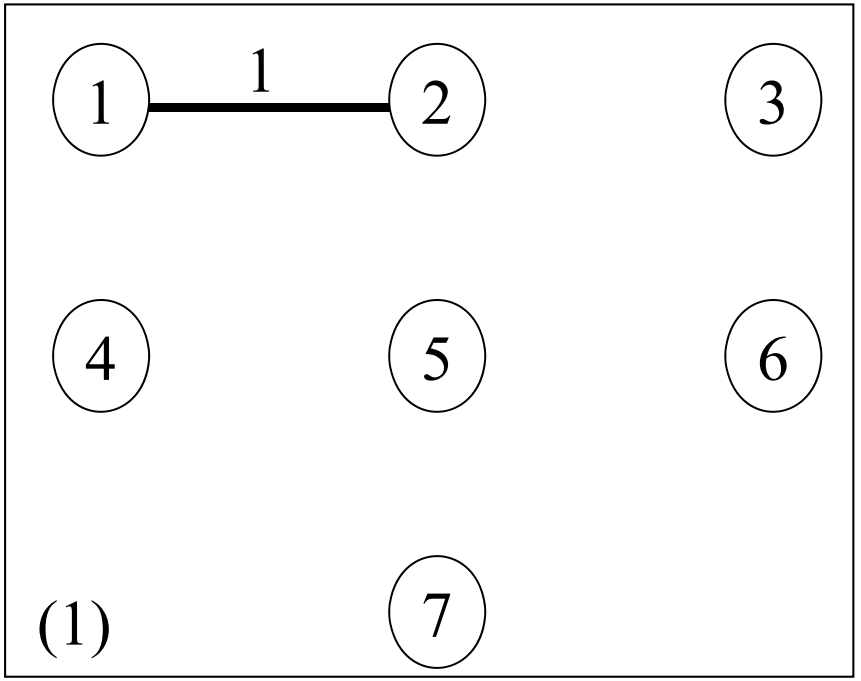

1
2
3
4
5
6
7
1
2
4
6
4
5
6
3
8
4
7
3
入力

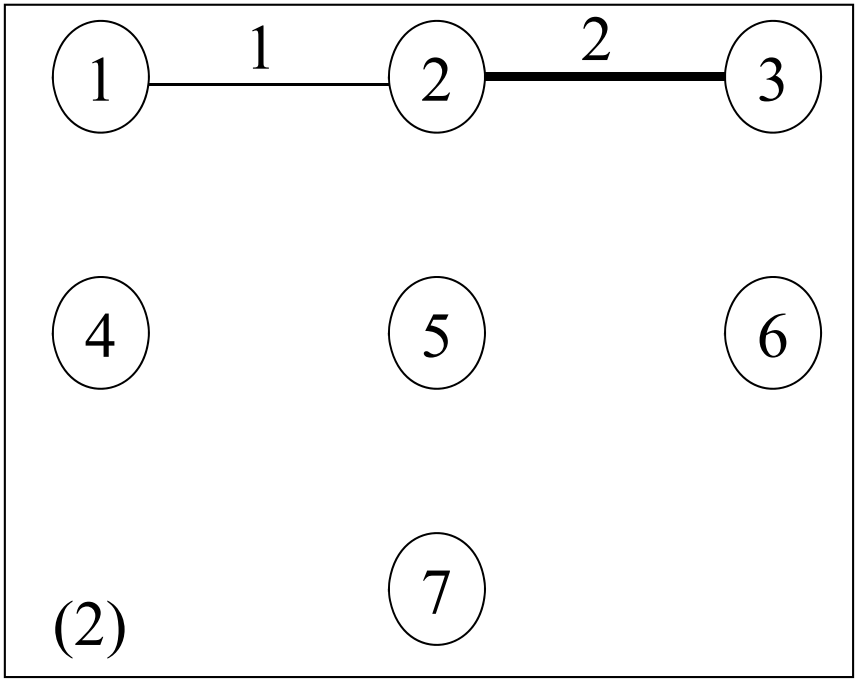

1
2
3
4
5
6
7
1
(1)

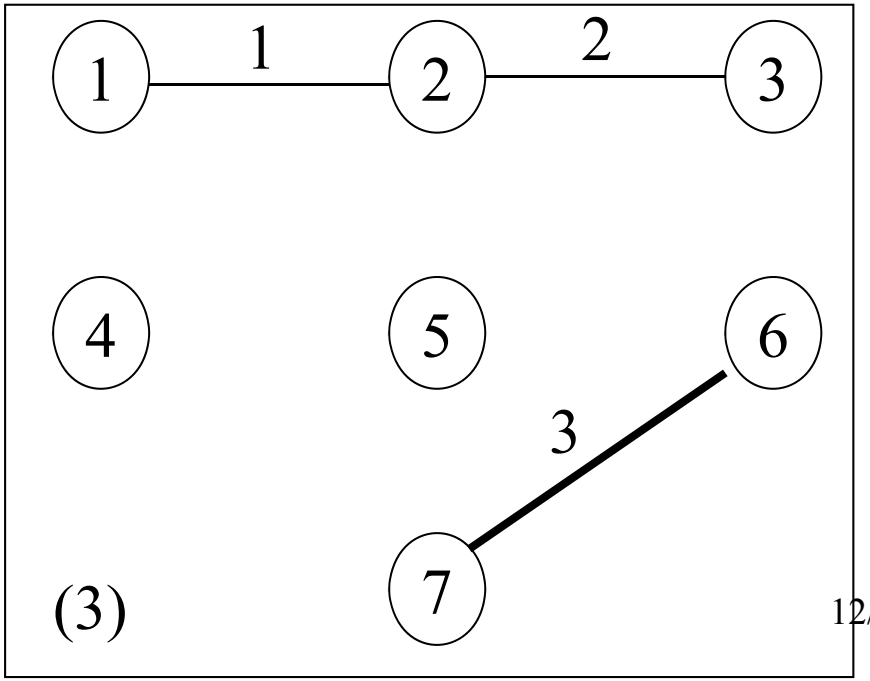

1
2
3
4
5
6
7
1
2
(2)

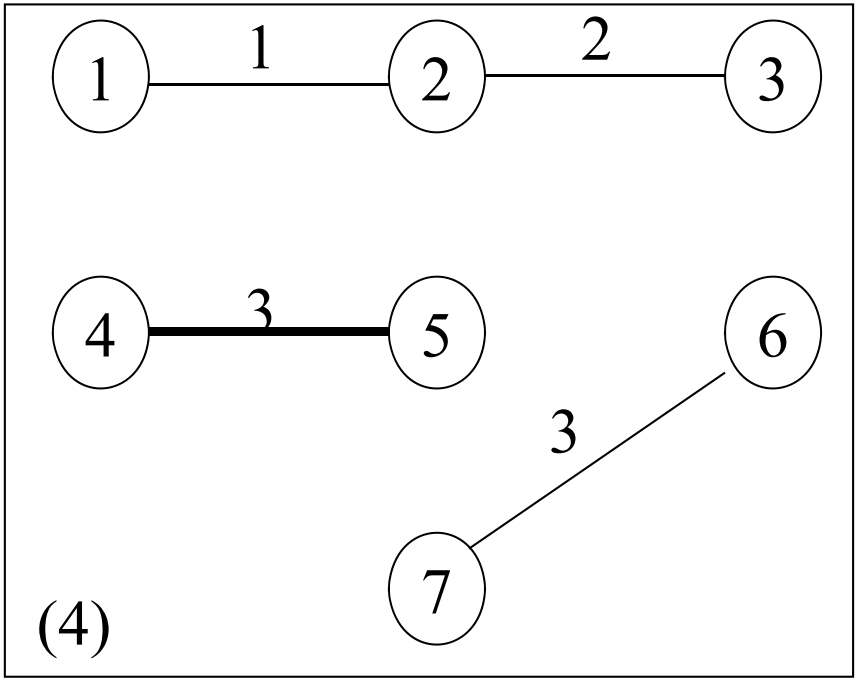

1
2
3
4
5
6
7
1
2
3
(3)

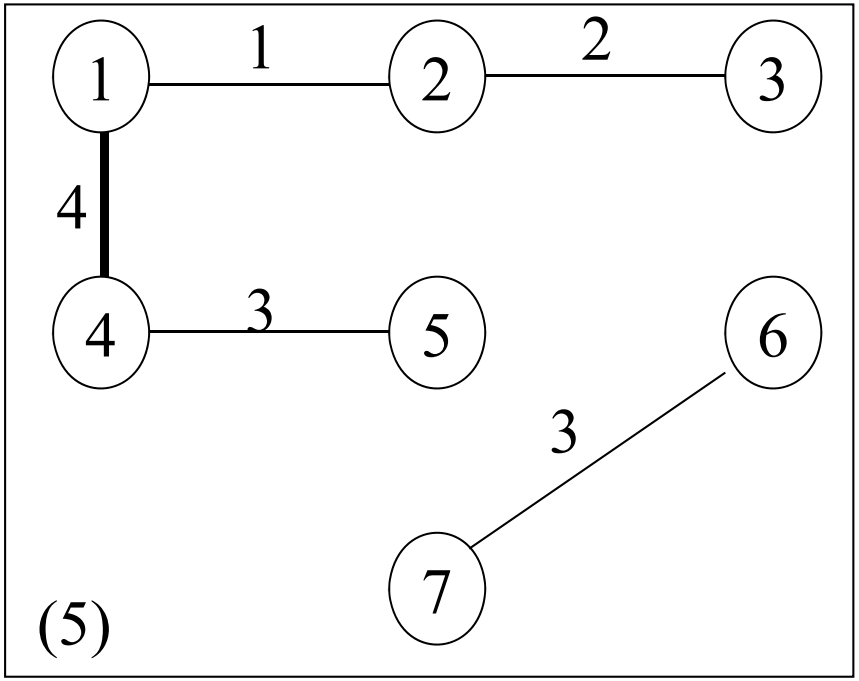

1
2
3
4
5
6
7
1
2
4
6
4
5
6
3
8
4
7
3
input graph

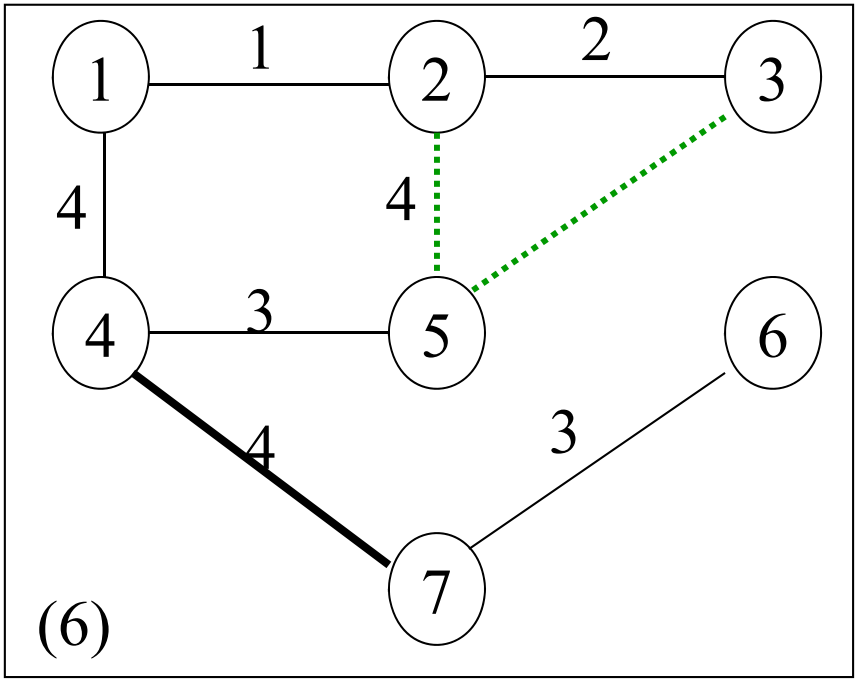

1
2
3
4
5
6
7
1
(1)

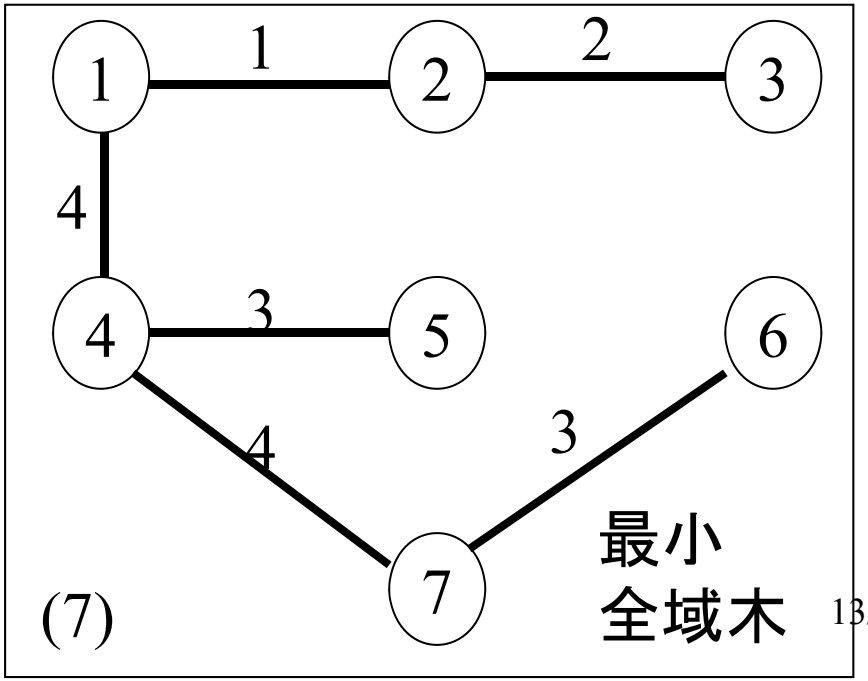

1
2
3
4
5
6
7
1
2
(2)

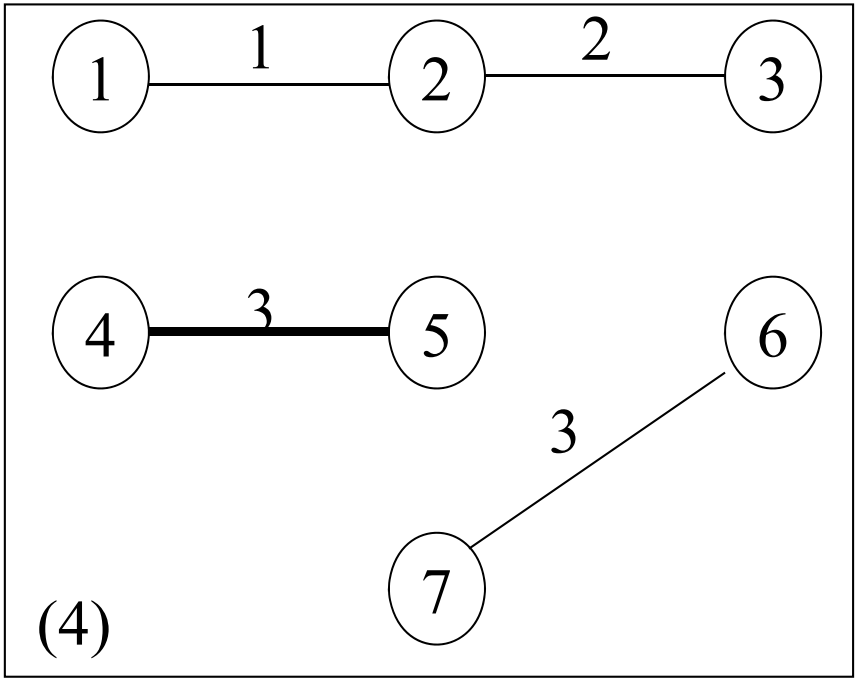

1
2
3
4
5
6
7
1
2
3
(3)

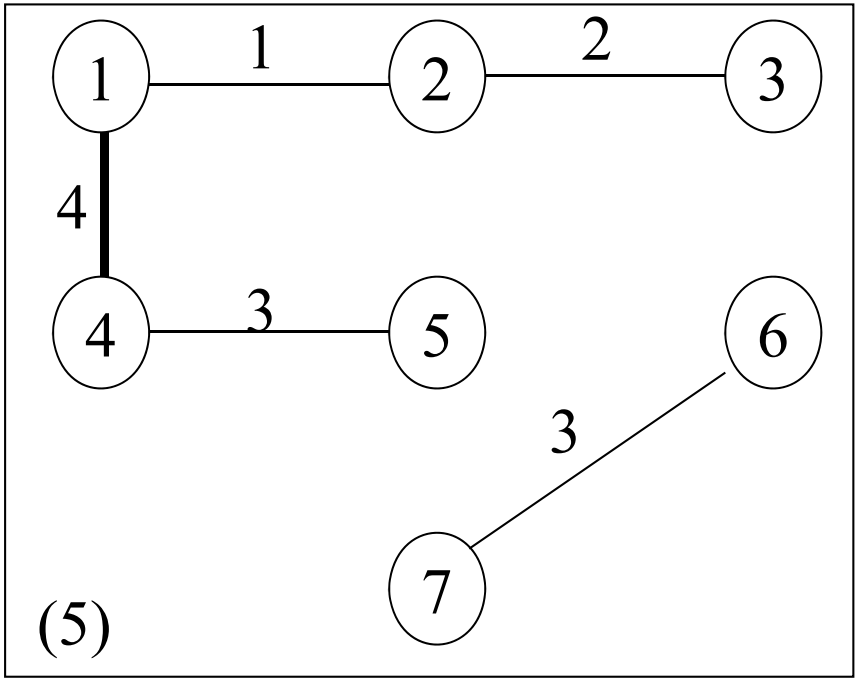

1
2
3
4
5
6
7
1
2
3
3
3
(4)

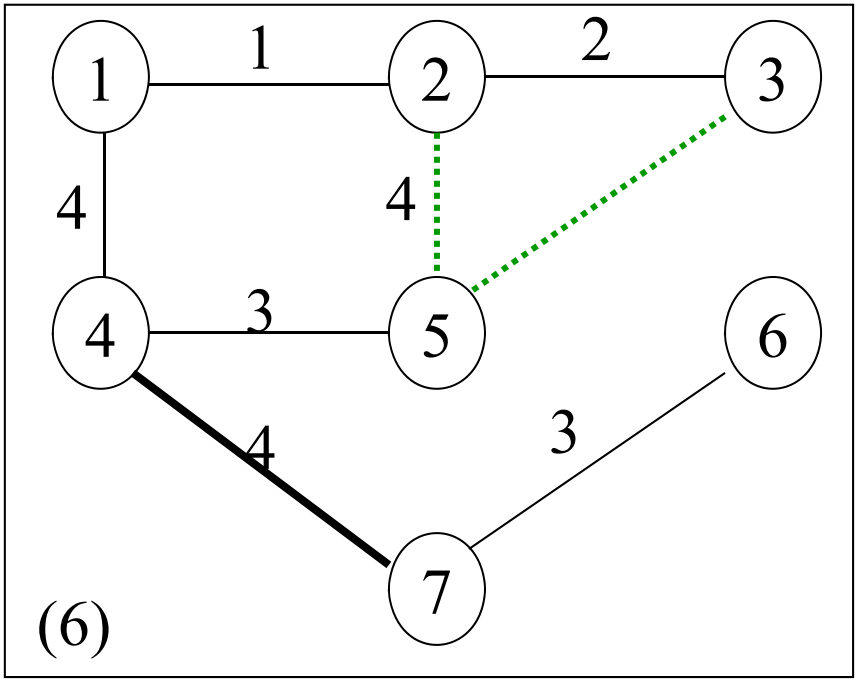

1
2
3
4
5
6
7
1
2
4
3
3
(5)

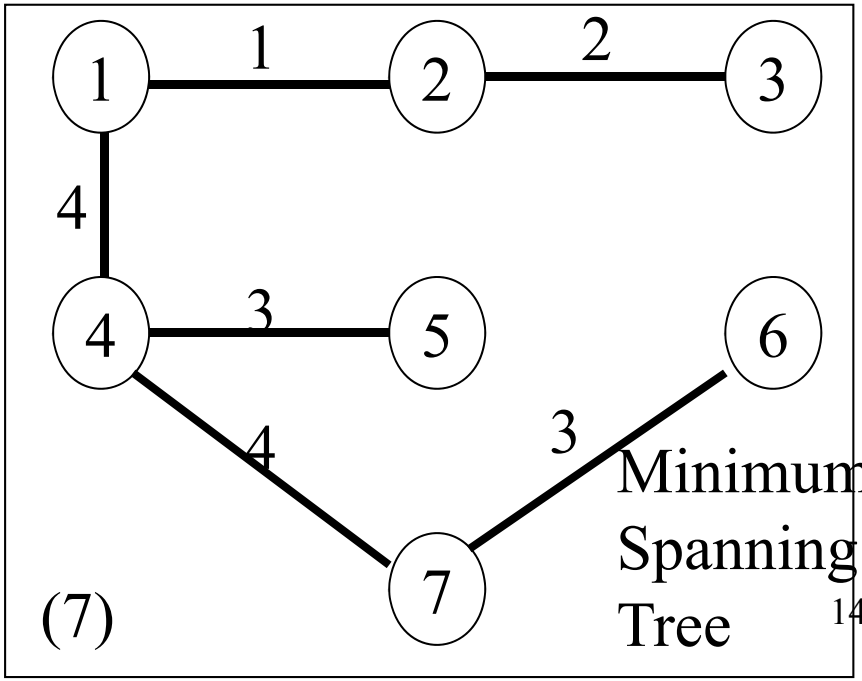

1
2
3
4
5
6
7
1
2
4
4
3
4
3
(6)


1
2
3
4
5
6
7
1
2
4
3
4
3
(7)
最小
全域木


1
2
3
4
5
6
7
1
2
3
3
3
(4)


1
2
3
4
5
6
7
1
2
4
3
3
(5)


1
2
3
4
5
6
7
1
2
4
4
3
4
3
(6)


1
2
3
4
5
6
7
1
2
4
3
4
3
(7)
Minimum
Spanning
Tree

アルゴリズムP12-A0: (Kruskalのアルゴリズム)

```
T = 空集合;
while(Eは空でない){
  Eの中から重み最小の辺eを選び, Eからeを取り除く;
  Tに辺eを加えてできるグラフがサイクルを含まなければ,
  Tにeを加える;
}
辺集合Tによって定まるグラフ(木)を最小全域木として出力;
```

アルゴリズムP12-A0によって求まるグラフは必ず木である.

理由:サイクルが生じないように辺を加えているから.

このような貪欲な方法で最適な木が求まっているだろうか？

=>アルゴリズムの正しさの検証

練習問題:頂点数12以上のグラフについてKruskalのアルゴリズムの動作を示せ.

Algorithm P12-A0: (Kruskal's algorithm)

```
T = empty set;
while(E is not empty){
  Choose an edge e of smallest weight, and remove it from E;
  Add the edge e to T if the graph obtained by adding e to T does
  not contain any cycle;
}
Output the graph (tree) determined by the edge set T as MST;
```

The graph obtained by Algorithm P12-A0 is always a tree.

Why? Edges are added not to generate any cycle.

Does this greedy algorithm find an optimal tree?

=>Verification of the correctness of the algorithm

Exercise: Show behavior of the Kruskal's algorithm for a graph with more than 11 vertices.

アルゴリズムの正しさ

辺の本数に関する帰納法

アルゴリズムで求まるTがk本の辺からなるとき，
「Tはk本の辺からなるGの部分グラフで，重み最小の木になっている（閉路を含まない）」

k=1のとき，重み最小の辺を選んでいるから，明らかに成立．

kまで成り立つとして，k+1のときを考える．

T’： k本の辺からなる閉路を含まないGの部分グラフで
重み最小のもの

e: T’に付加しても閉路を生じない辺の中で重み最小の辺

T’に辺eを付加するとk+1本の辺からなる部分グラフを得る：

閉路を含まないように辺を選んでいる

重みも最小である

∵どの辺も他の辺で置きかえると，
閉路を生じるか，辺の重みが増加する．

Correctness of Algorithm

Induction on the number of edges

When a tree T obtained by the algorithm has k edges,
"T is a subgraph of G consisting of k edges which is a tree of minimum weight (without any cycle)"

For k=1 it holds since it contains the edge of the minimum weight.

Assuming that it holds for 1, ... , k, consider the case for k+1.

T' : an acyclic subgraph of G containing k edges that has the smallest weight

e: an edge of smallest weight such that its addition to T' causes no cycle.

Adding the edge e to T', we have a subgraph with k+1 edges.

The edge is selected so that no cycle is generated.

Its weight is minimum.

∵ Replacing any edge with some other edge, some cycle is generated or the edge weight increases.

アルゴリズムの実現

1. 辺を重みの昇順に取り出す
   最初にすべての辺を重みの昇順にソートしておけばよい.
2. 辺を付加したとき閉路を生じるかどうかの判定
   辺(u, v)を付加するごとに頂点uとvが属する連結成分を統合
   →uとvが既に同じ連結成分に属していれば閉路を生じる

では，どのように連結成分を管理するか？
→union-find木のデータ構造

頂点の連結成分を木の形で管理

union(u, v)操作
uを含む木とvを含む木を
1つに統合する.

find(u)操作
uを含む木の根を答える

初期化操作
各頂点uについて，uを唯一の頂点とする木(根はu)で表す.

Implementation of Algorithm

1. Take edges in the increasing order of their weights.
   It suffices to sort all edges in the order of weights.
2. Determine whether addition of a new edge causes a cycle.
   Whenever we add an edge (u, v), we merge the connected components of u and w into one.
   →If u and v belong to the same component then a cycle arises.
   Then, how can we maintain connected components?
   →union-find tree data structure

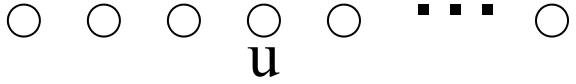
Connected components of vertices are maintained in a tree.

union(u, v) operation
Merge the tree containing u with that containing v.

find(u) operation
answer the root of a tree containing u

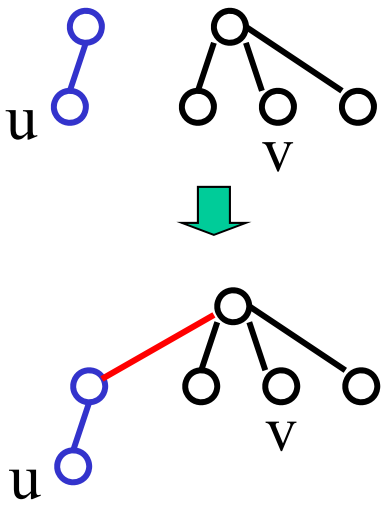
Initialization
For each vertex u, we have a tree with u as its only vertex.

初期化操作
各頂点uについて，uを唯一の頂点とする木(根はu)で表す．



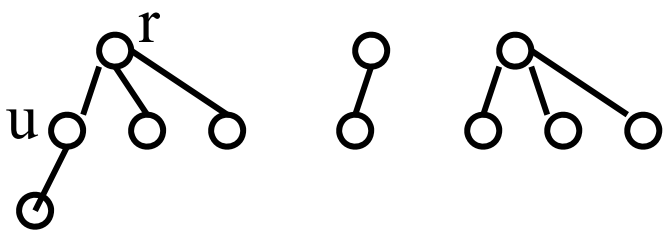
最初はすべて孤立点

union(u, v)操作
uを含む木とvを含む木を
1つに統合する．



uを含む木の根とvを含む木の根を辺で結ぶ．

find(u)操作
uを含む木の根を答える



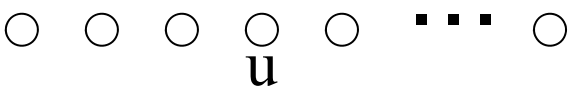
uから木の辺を根に向けて辿り，根を答える．

計算時間の解析は易しくないが，頂点数をn，辺数をmとすると，O(mα(n))時間にできる．

α(n):アッカーマン関数の逆関数

Initialization

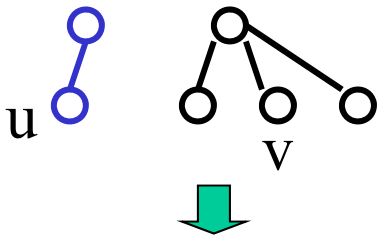
For each vertex u, represent by u the tree with u as a unique vertex.



Initially, all the vertices are isolated

union(u, v) operation

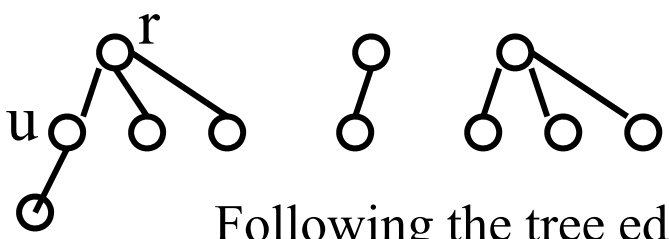
Merge the tree containing u with that containing v.



Connect the root of the tree containing u with the root of the tree containing v

find(u) operation

answer the root of a tree containing u



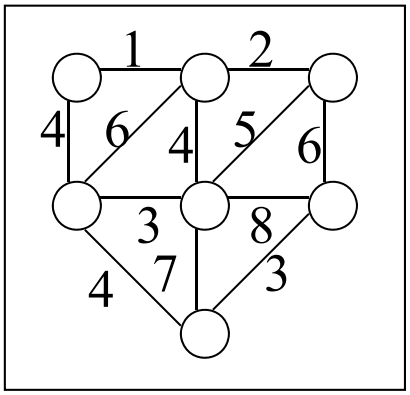
Following the tree edges from u toward the root, return the root.

Analysis of computation time is not easy, but it is shown to be $O(m\alpha(n))$ with n vertices and m edges.

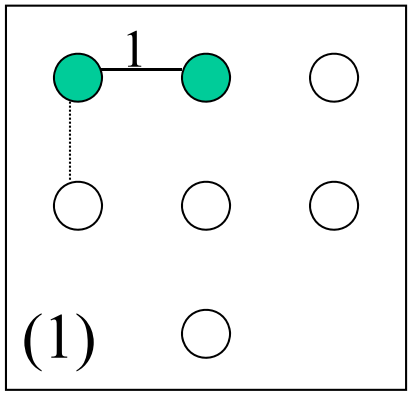
$\alpha(n)$:Inverse of the Ackermann function

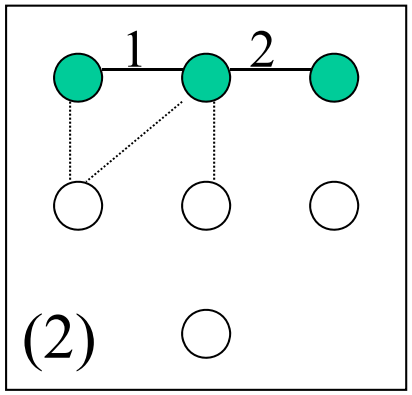
Primのアルゴリズム

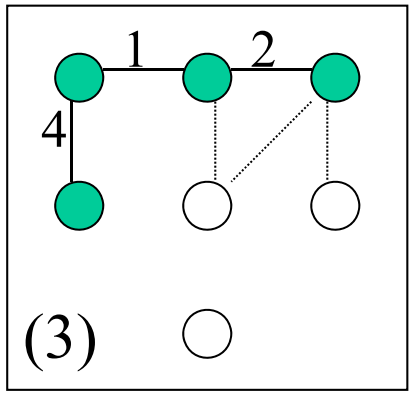
閉路を生じるかどうかを判定するのではなく，閉路が生じないように辺を増やしていく方法.
木Tを1つの頂点だけからなるものから始めて，Tの頂点とTの外の頂点を結ぶ辺の中で重み最小のものを求め，Tに加える.

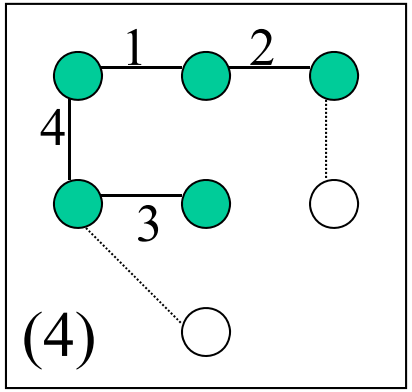


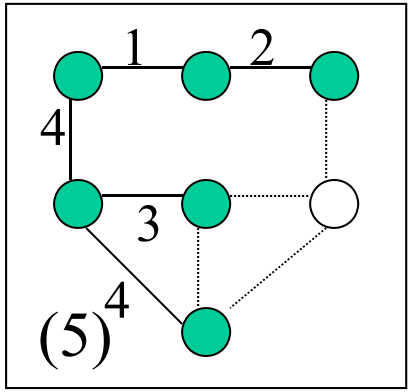
グラフG

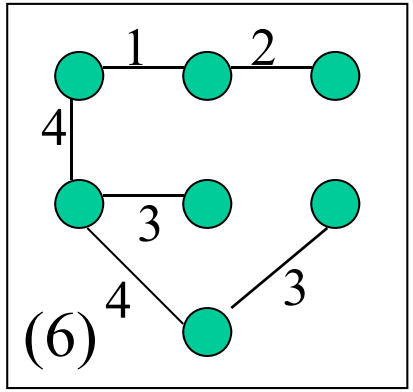






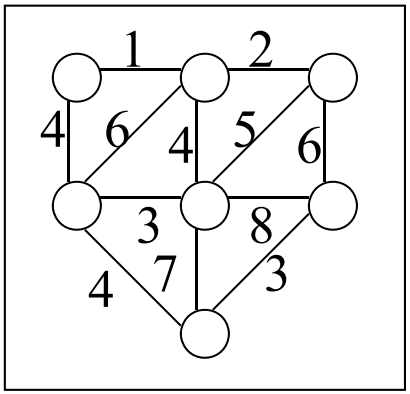




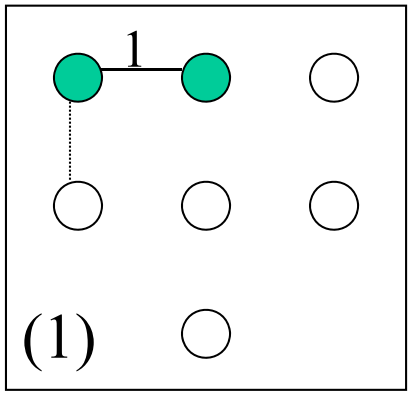


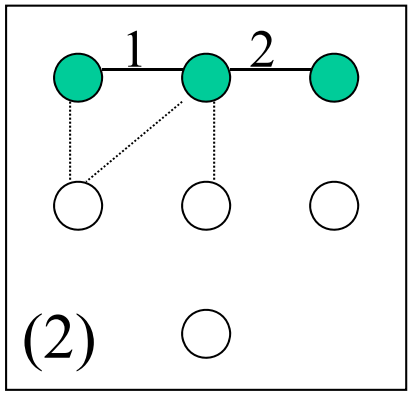
Prim's Algorithm

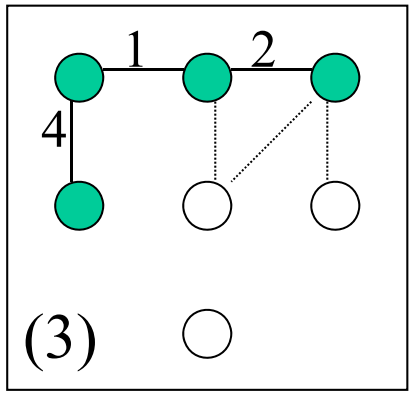
It is an algorithm that does not check but adding edges so that no cycle is generated. Starting T from a tree containing only one vertex, we find an edge of smallest weight connecting vertices of T with those outside T and add it to T.

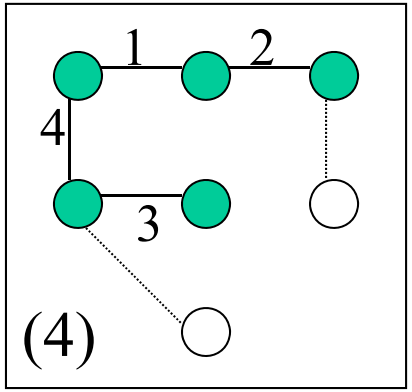


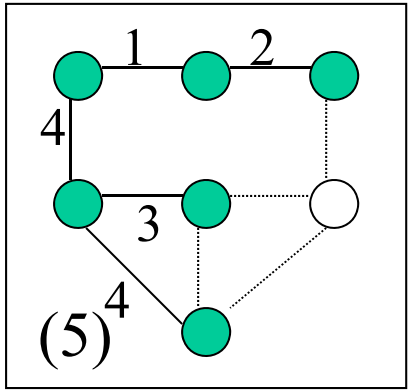
graph G

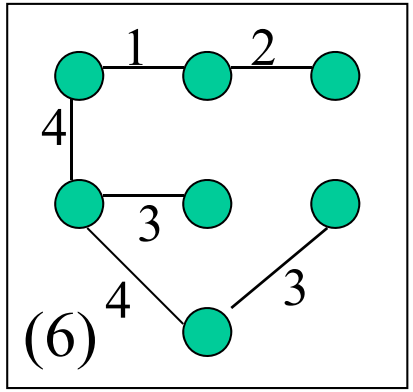












アルゴリズムP12-A1:(Primのアルゴリズム)

```
Tを空集合;
 任意の頂点uを選び，B={u}と初期化；
        // Bは選ばれた頂点の集合．すべての頂点を選べば終了
 while(V-Bが空でない){
   V-Bに属する頂点uとBに属する頂点vを結ぶ辺(u,v)の中で
   重みが最小のものを選ぶ;
      //つまり，選ばれた頂点と選ばれていない頂点を結ぶ辺
   Tに辺(u,v)を加える;
   Bに頂点uを加える;
}
 辺集合Tによって定まるグラフ（木）を最小全域木として出力;
```

Algorithm P12-A1:(Prim's algorithm)

```
  Let T be an empty set;
   Choose any vertex u and initialize B as B={u};
   // B is a set of selected vertices.  Stop when all vertices are selected.
while(V-B is not empty){
     Choose an edge (u, v) of smallest weight connecting a vertex u in
     V-B and a vertex v in B;
     //i.e., an edge between selected vertices and unselected vertices
     Add the edge (u, v) into T;
     Add the vertex u to B;
}
Output the graph (tree) determined by the set T of edges as a minimum
spanning tree;
```

貪欲法の原理

貪欲法によって解ける最適化問題の特徴づけ

1. 貪欲選択性(greedy-choice property)

最適解が局所的に最適な解を繰り返し選ぶことで得られるという性質.

2. 部分構造の最適性

最適解が部分問題に対する最適解を含むという性質

0-1ナップサック問題と一般化ナップサック問題

入力:n個の品物の価値と体積，およびナップサックの容量C

出力:ナップサックに収容可能な品物の価値の総和の最大値

- 0-1ナップサック問題では1つの品物を分割することはできず，ナップサックに入れるかどうかを決めなければならない.
- 一般化ナップサック問題では，1つの品物を任意に分割可能.

どちらの問題が難しいか？

Principle of Greedy Algorithms

Characterizing optimization problems solved by greedy algorithms

1. Greedy-choice property

A property that an optimal solution is obtained by successive selection of locally optimal solutions.

2. Optimal Substructure

A property that an optimal solution includes an optimal solution to a subproblem.

0-1 Knapsack Problem and Generalized Knapsack Problem

Input: values and volume of n items, and capacity C of knapsack.

Output: Maximum sum of values of items that be put into knapsack.

Any item cannot be decomposed in the 0-1 knapsack problem.

If we choose an item, we have to put the whole item into knapsack.

In the generalized knapsack problem, we can decompose any item into pieces. Which problem is harder?

例：品物1=(60,000円，10m^3), 品物2=(100,000円，20m^3),
品物3=(120,000円，30m^3), 容量C=50 m^3

単位体積あたりの価値を求めると，
品物1 = 6000, 品物2=5000, 品物3=4000

0-1ナップサック問題では

・品物1，品物2の順に取ると，品物3は容量の関係でとれないから，価値の総和は160,000円になる．

・品物1，品物3の順にとると，価値の合計は180,000円，
品物2と3を取ると，価値の合計は220,000円となって最大．

・様々な組合せがあるので，難しい問題である（NP完全問題）

一般化ナップサック問題では

・単位堆積あたりの価値の高い順に，品物1，品物2の順に取り，残りの容量で品物3を取ると，価値の総和は

60,000+100,000+120,000*20/30=240,000円

となり，最大の値となる．
つまり，貪欲法で最適解が求まる．

Example : Item1=(60,000yen, $10m^3$), Item2=(100,000yen, $20m^3$), Item3=(120,000yen, $30m^3$), Capacity C=50 m^3

Calculating the values per unit volume,

Item1 = 6000, Item2=5000, Item3=4000

0-1 Knapsack Problem

- Choosing Item1 and Item2 in order, we cannot take Item3 due to capacity constraint. So, the total value becomes 160,000 yen.
- Choosing Item1 and Item3 in order, the total value is 180,000 yen, and choosing Item2 and Item3 then it is 220,000 yen, which is highest.
- Since there are so many combinations, it is a hard problem. NP complete problem.

Generalized Knapsack Problem

- Taking the items in the decreasing order of their values, i.e., Items1 and Items, and taking Item3 for the remaining capacity. Then, the total value amounts to

  60,000+100,000+120,000*20/30=240,000 yen,

that is the highest value. That is, the greedy algorithm can find an optimal solution.

一般化ナップサック問題の性質
部分構造の最適性

・最適解が部分問題に対する最適解を含むという性質
・最初に単位体積あたりの価値が最大である品物1を
全部($10m^3$) だけ取る. この$10m^3$の分を他のどの品物で
置き換えても価値は下がるから, この選択は最適である.
つまり, 残りの$40m^3$を品物2と品物3だけで詰めるという
部分問題の最適解に, 品物1の$10m^3$の分を加えれば
全体の最適解が求まる.

問題P13:(一般化ナップサック問題)
n個の品物の価値と体積, およびナップサックの容量Cが与えられたとき, ナップサックに収容可能な品物の価値の総和を最大にするには, それぞれの品物をそれぞれどれだけ選べばよいか？

Property of the Generalized Knapsack Problem

Optimal Substructure

- Property that an optimal solution contains an optimal solution to a subproblem.
- First, we take the whole of the Item1 ($10m^3$) of the highest value per unit volume. Exchanging this $10m^3$ with any other item decreases the total value, and so this selection is optimal.
  That is, the globally optimal solution is obtained by filling the remaining $40m^3$ with Item2 and Item3 together with $10m^3$ filled by Item1.

Problem P13:(Generalized Knapsack Problem)
Given values and volume of n items and the capacity C of knapsack, how much of each item should we choose so that the total value of items to be put into knapsack is maximized?

アルゴリズムP13-A0:（貪欲法）

n個の品物の価値と体積を$(v_0, w_0), \ldots (v_{n-1}, w_{n-1})$とする.
ナップサックの容量をCとする.
最初に，それぞれの品物の単位体積あたりの価値v_i/w_iを求める.
n個の品物を単位体積あたりの価値の降順にソートする.

$v_0/w_0 \geqq v_1/w_1 \geqq \cdots \geqq v_{n-1}/w_{n-1}$とする.

上のソート順に従って品物をナップサックに入れていく.
ただし，ナップサックの容量Cを超過すれば，超過分を削除して終り.

```
i=0; sum=0;
while(i<n && sum+w_i≦C){
do{
  品物iをナップサックに入れる;  i=i+1; sum=sum+w_i;
}
最後に品物iをC-sum分だけ入れる;
```

計算時間はO(n log n) <=ソート処理にO(n log n)必要

Algorithm P13-A0:（Greedy Algorithm）

Let values and volume of n items be (v_0, w_0), ... (v_{n-1}, w_{n-1}).

Let C be the capacity of knapsack.

First, find unit value v_i/w_i of each item per unit volume.

Then, sort n items in the decreasing order of their unit values.

Assume that $v_0/w_0 \geqq v_1/w_1 \geqq \cdots \geqq v_{n-1}/w_{n-1}$

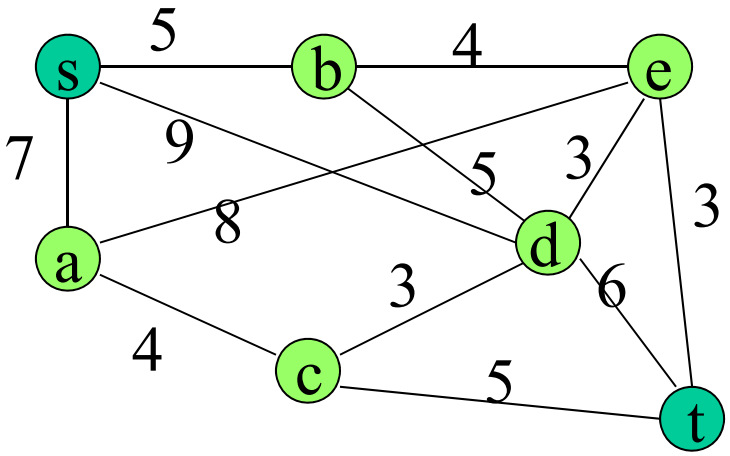
According to the sorted sequence above we put items into knapsack.

When the total volume exceeds the capacity C of knapsack, then we stop after removing the excessive portion.

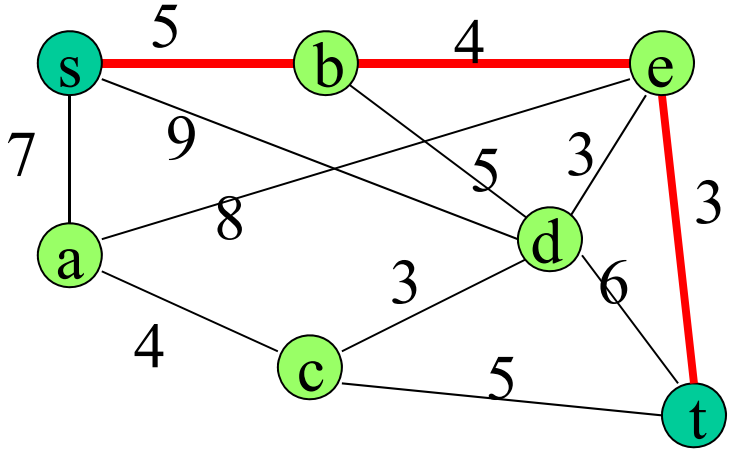
```
i=0; sum=0;
while(i<n && sum+w_i≦C){
do{
  Put the item i into knapsack;  i=i+1; sum=sum+w_i;
}
Finally, put the item i by C-sum;
```

Computation time is O(n log n) <=O(n log n) is required for sorting

問題P14:（最短経路問題）
n都市を結ぶ道路網が重みつきグラフの形で与えられているとき，
任意に2都市を結ぶ最短経路（重み最小の経路）を求めよ.

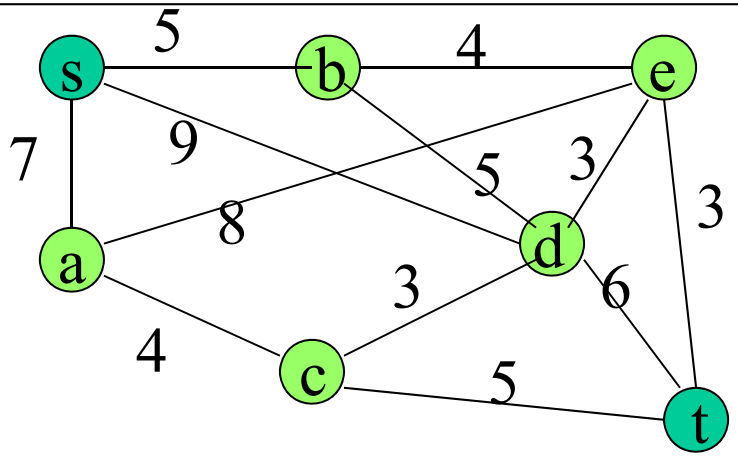


sからtへの最短経路？

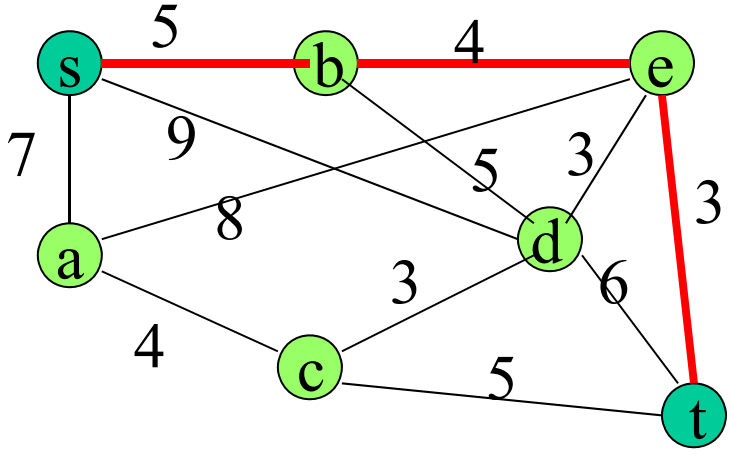


Problem P14:（Shortest Path Problem）

Given a road network connecting n cities as a weighted graph, find a shortest path (of minimum weight) between arbitrarily specified two cities.



Shortest path from s to t?



ダイクストラのアルゴリズム

始点sから終点tまでの最短経路だけでなく，sから他の全ての頂点への最短経路を求める．
D[v]: sから頂点vまでの最短経路の長さ
として，すべての頂点についてD[]の値を求める．

貪欲選択：

最初は

$D[s]=0$

$D[v]=\infty$ $(v \neq s)$

として始め，毎回D[v]の値が最小の頂点を選んで，その値を確定した後，頂点vに隣接する頂点への辺を調べて，隣接頂点へのより短い経路を調べる．

Dijkstra's Algorithm

Find a shortest path from a starting vertex s to every other vertex in addition to a shortest one from s to a terminating vertex t.
D[v]: length of a shortest path from s to a vertex v.
We want to calculate the values D[] for all the vertices.

Greedy choice:

Initially we have

D[s]=0

D[v]=∞ (v≠s).

Every time we choose a vertex v
to minimize the value D[v] and determine the value D[v].
Then, we examine its adjacent edges to find shorter paths
to its adjacent vertices.

アルゴリズムP14-A0:（ダイクストラ法）

C=s以外のすべての頂点からなる集合;
for すべての頂点v do D[v] = ∞;
D[s] = 0;
u = s;
do{
  for 頂点uに隣接するすべての頂点v do
    if D[u]+w(u,v) < D[v] then D[v] = D[u]+w(u,v);
  Cの中でD[]の値が最小の頂点を選び，uとする.
  Cからuを削除する.
} while(Cが空でない);

演習問題：ダイクストラのアルゴリズムの計算時間について他のテキストを調査せよ.

演習問題：アッカーマン関数について調査せよ.

Algorithm P14-A0: (Dijkstra's Algorithm)

```
C= a set of vertices except s;
for each vertex v do D[v] = ∞;
D[s] = 0;
u = s;
do{
   for each vertex v adjacent to vertex u do
      if D[u]+w(u,v) < D[v] then D[v] = D[u]+w(u,v);
   Choose a vertex u to minimize D[] among C;
   Remove u from C;
} while(C is not empty);
```

Exercise: Examine textbooks for the computation time of the Dijkstra's algorithm.

Exercise: Examine the Ackermann function.