

Chapter 8

The Greedy Approach

贪心算法

顾名思义，贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的**局部最优**选择。当然，希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路径问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

- 0-1 背包问题：

给定 n 种物品和一个背包。物品 i 的重量是 W_i ，其价值为 V_i ，背包的容量为 C 。应如何选择装入背包的物品，使得装入背包中物品的总价值最大？

在选择装入背包的物品时，对每种物品 i 只有2种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分的物品 i 。

- 背包问题：

与0-1背包问题类似，所不同的是在选择物品 i 装入背包时，**可以选择物品 i 的一部分**，而不一定要全部装入背包， $1 \leq i \leq n$ 。

这2类问题都具有**最优子结构**性质，极为相似，但背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。

用贪心算法解背包问题的基本步骤：

首先计算每种物品单位重量的价值 V_i/W_i ，然后，依贪心选择策略，将尽可能多的**单位重量价值最高**的物品装入背包。若将这种物品全部装入背包后，背包内的物品总重量未超过 C ，则选择单位重量价值次高的物品并尽可能多地装入背包。依此策略一直地进行下去，直到背包装满为止。

对于**0-1背包问题**，贪心选择之所以不能得到最优解是因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此就导出许多互相重叠的子问题。这正是该问题可用**动态规划算法**求解的另一重要特征。

实际上也是如此，动态规划算法的确可以有效地解0-1背包问题。

单源最短路径

给定带权有向图 $G=(V,E)$ ，其中每条边的权是非负实数。另外，还给定 V 中的一个顶点，称为**源**。现在要计算从源到所有其他各顶点的**最短路长度**。这里路的长度是指路上各边权之和。这个问题通常称为**单源最短路径问题**。

1. 算法基本思想

Dijkstra算法是解单源最短路径问题的贪心算法。

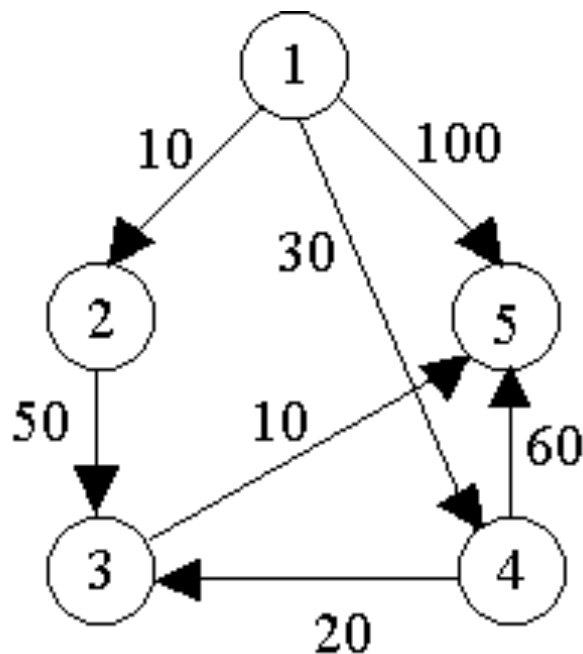
单源最短路径

其**基本思想**是，设置顶点集合 S 并不断地作**贪心选择**来扩充这个集合。一个顶点属于集合 S 当且仅当从源到该顶点的最短路径长度已知。

初始时， S 中仅含有源。设 u 是 G 的某一个顶点，把从源到 u 且中间只经过 S 中顶点的路称为从源到 u 的特殊路径，并用数组 $dist$ 记录当前每个顶点所对应的最短特殊路径长度。**Dijkstra**算法每次从 $V-S$ 中取出具有最短特殊路长度的顶点 u ，将 u 添加到 S 中，同时对数组 $dist$ 作必要的修改。一旦 S 包含了所有 V 中顶点， $dist$ 就记录了从源到所有其他顶点之间的最短路径长度。

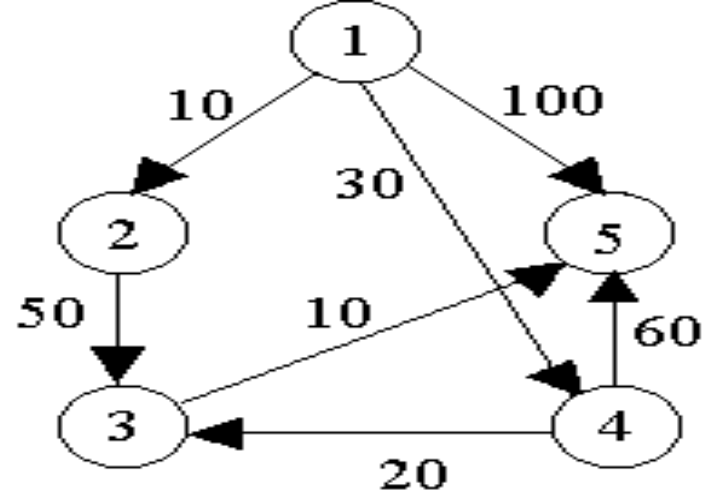
单源最短路径

例如，对右图中的有向图，应用Dijkstra算法计算从源顶点1到其他顶点间最短路径的过程列在下页的表中。



单源最短路径

Dijkstra算法的迭代过程：

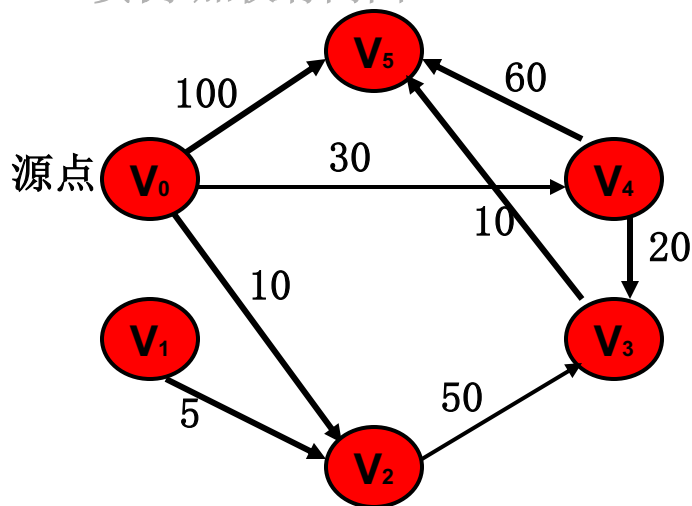


迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

最短路径

1、从某个源点到其余各顶点的最短路径（单源最短路径问题）

• 实例:加权有向图



加权邻接矩阵:

	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

源点	终点	最短路径	路径长度
V_0	V_1	无	
	V_2	(V_0, V_2)	10
	V_3	(V_0, V_4, V_3)	50
	V_4	(V_0, V_4)	30
	V_5	(V_0, V_4, V_3, V_5)	60

1. $X \leftarrow \{1\}; Y \leftarrow V - \{1\}$
2. For each vertex $v \in Y$ if there is an edge from 1 to v then let $\lambda[v]$ (the label of v) be the length of that edge; otherwise let $\lambda[v] = \infty$. Let $\lambda[1] = 0$
3. while $Y \neq \{\}$
4. Let $y \in Y$ be such that $\lambda[y]$ is minimum
5. move y from Y to X
6. update the labels of those vertices in Y that are adjacent to y
7. end while

Algorithm 8.1 DIJKSTRA

Input: A weighted directed graph $G=(V,E)$, where $V=\{1,...,n\}$

Output: The distance from vertex 1 to every other vertex in G

1. $X=\{1\}$; $Y \leftarrow V-\{1\}$; $\lambda[1] \leftarrow 0$
2. for $y \leftarrow 2$ to n
3. if y is adjacent to 1 then $\lambda[y] \leftarrow \text{length}[1,y]$
4. else $\lambda[y] \leftarrow \infty$
5. end if
6. end for
7. for $j \leftarrow 1$ to $n-1$
8. Let $y \in Y$ be such that $\lambda[y]$ is minimum
9. $X \leftarrow X \cup \{y\}$ {add vertex y to X }
10. $Y \leftarrow Y - \{y\}$ {delete vertex y from Y }
11. for each edge (y,w)
12. if $w \in Y$ and $\lambda[y] + \text{length}[y,w] < \lambda[w]$ then
13. $\lambda[w] \leftarrow \lambda[y] + \text{length}[y,w]$
14. end if
15. end for

Minimum cost spanning trees

Definition 8.1: Let $G=(V,E)$ be a connected undirected graph with weights on its edges. A spanning tree (V,T) of G is a subgraph of G that is a tree. If G is weighted and the sum of the weights of the edges in T is minimum, then (V,T) is called a minimum cost spanning tree or simply a minimum spanning tree.

Greedy methods for MST

Q: How to grow an MST by adding one edges at a time?

A: The algorithm manages a set of edges A , maintaining the following loop invariant:

Prior to each iteration, A is a subset of some MST

That is, at each step we determine an edge (u,v) such that

$A \cup \{(u,v)\}$ is still a subset of a MST and (u,v) is called a safe edge for A

Generic MST algorithm

Given a connected, undirected graph $G=(V,E)$ with weights on its edges

GENERIC-MST(G,w)

1. $A \leftarrow \phi$
2. while A does not form a spanning tree
3. find an edge (u,v) that is safe for A
4. $A \leftarrow A \cup \{(u,v)\}$
5. end while
6. return A

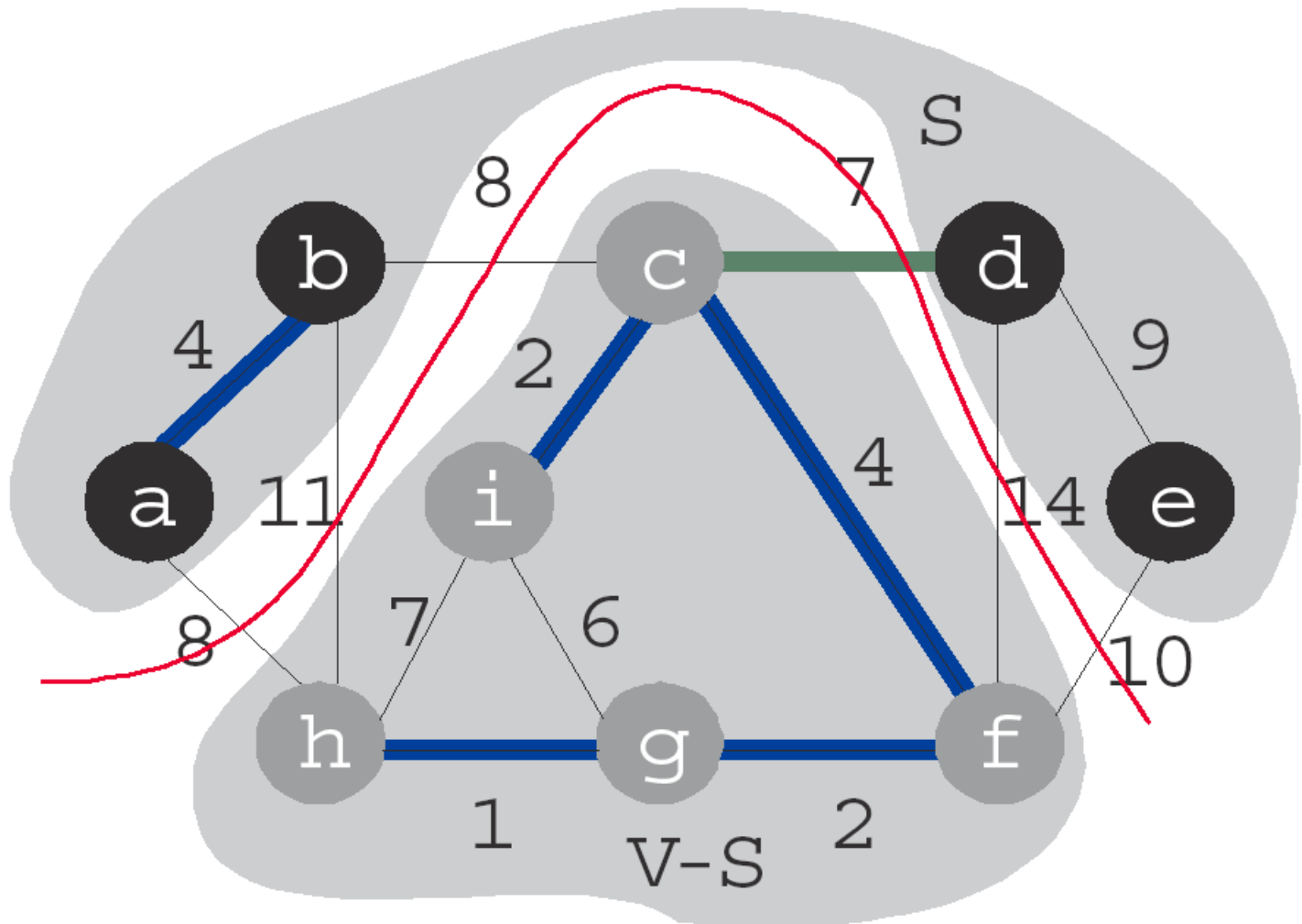
In the followings, we will provide a rule for recognizing safe edges, and the two algorithms use the rule efficiently

Cut and light edge

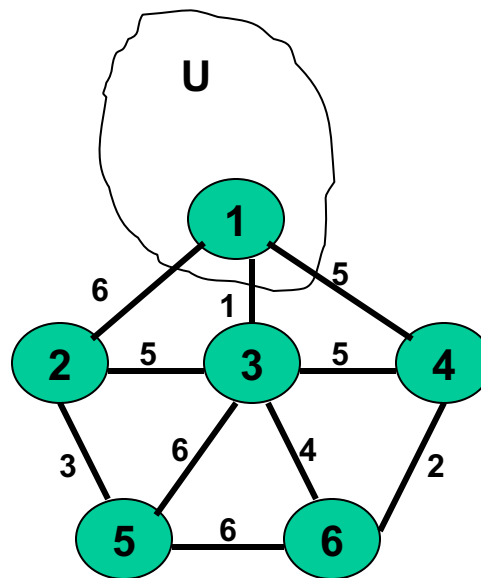
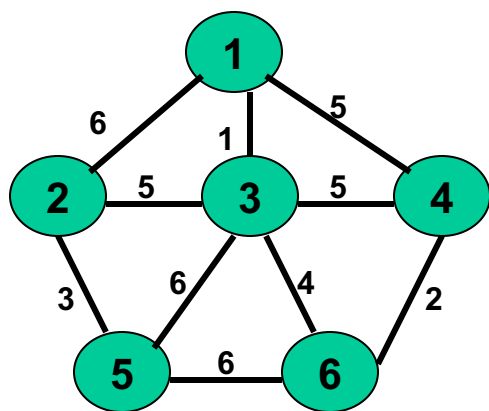
Definition: A cut $(S, V-S)$ of an undirected graph $G=(V, E)$ is a partition of V . An edge $(u, v) \in E$ crosses the cut iff one of its endpoints is in S and the other is in $V-S$. We say that a cut respects a set A of edges if no edge in A crosses the cut.

Definition: An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut

Example of cut and light edge

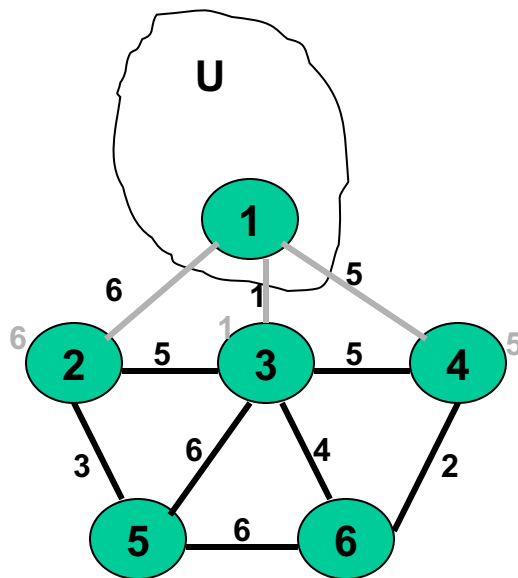
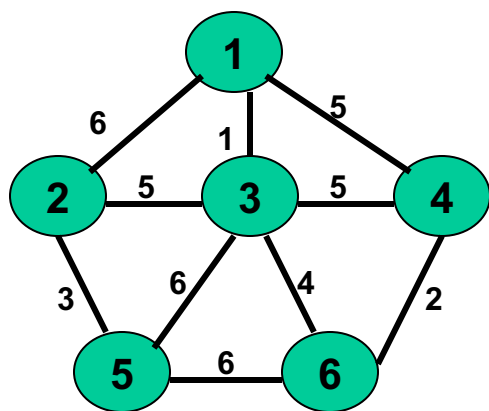


Prim 算法的实例



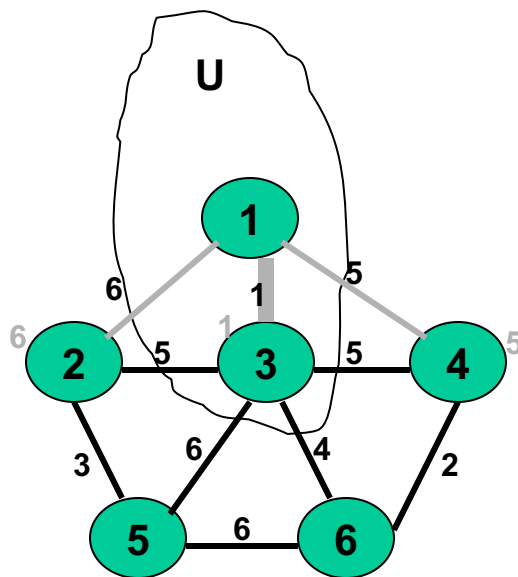
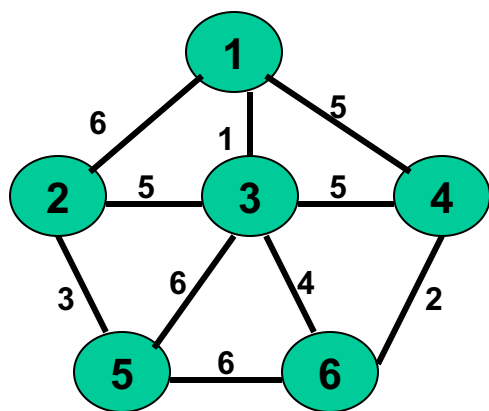
最小代价生成树
的生成过程

Prim 算法的实例



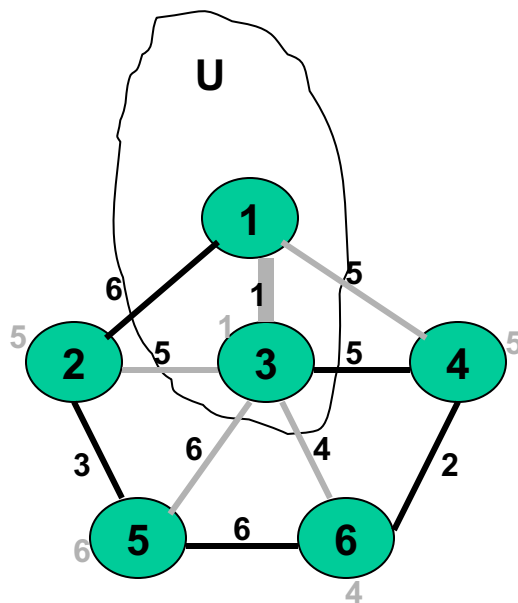
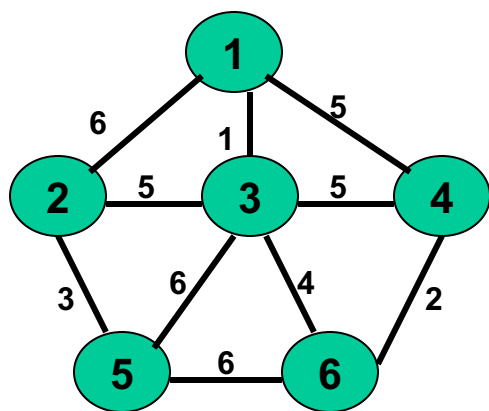
最小代价生成树
的生成过程

Prim 算法的实例



最小代价生成树
的生成过程

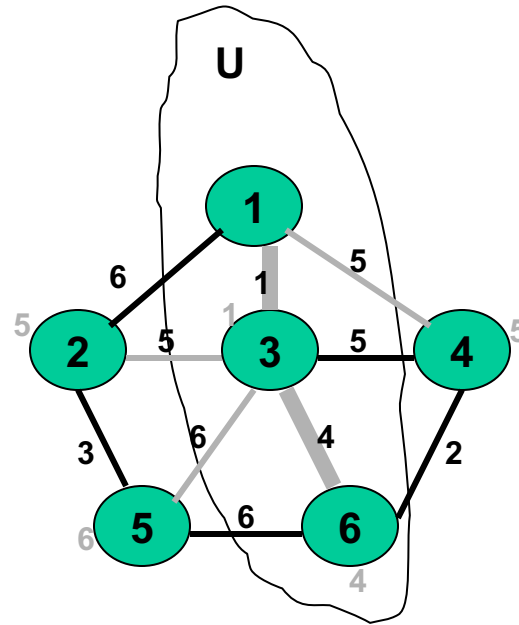
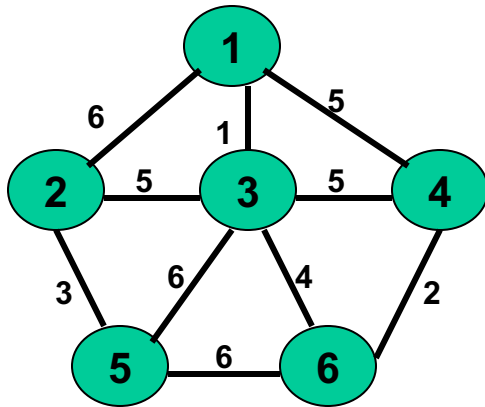
Prim 算法的实例



最小代价生成树
的生成过程

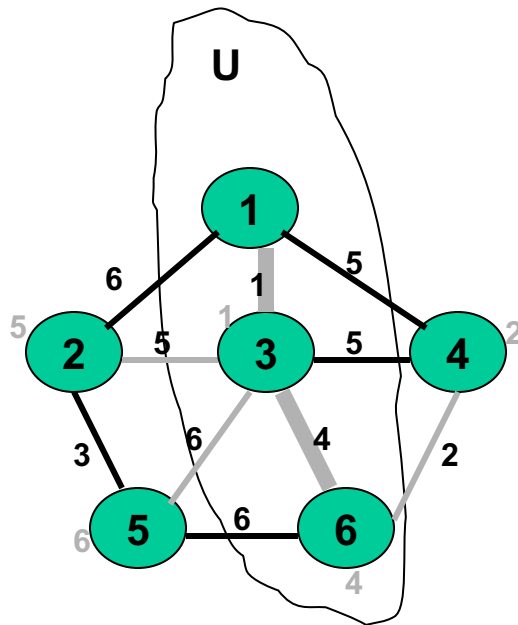
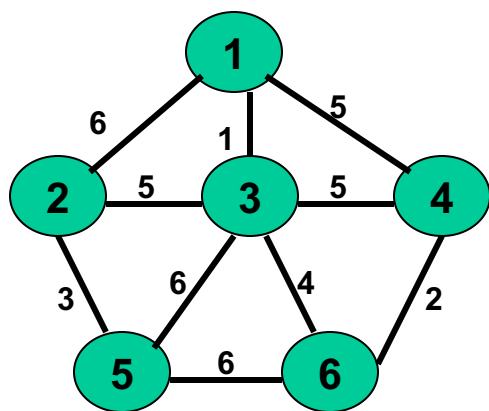
Prim 算法的实例

- Prim 算法的实例



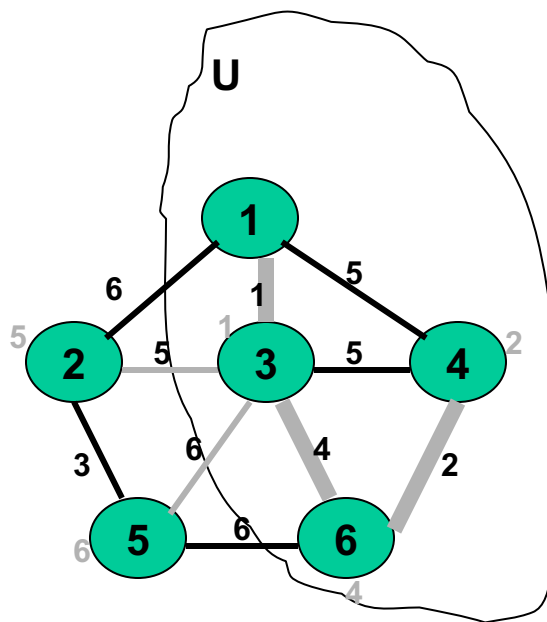
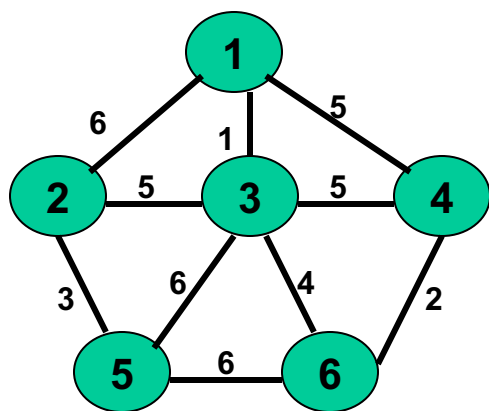
最小代价生成树
的生成过程

Prim 算法的实例



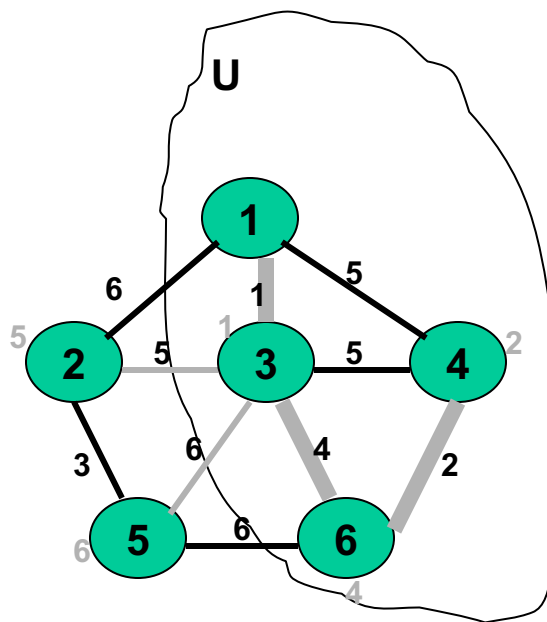
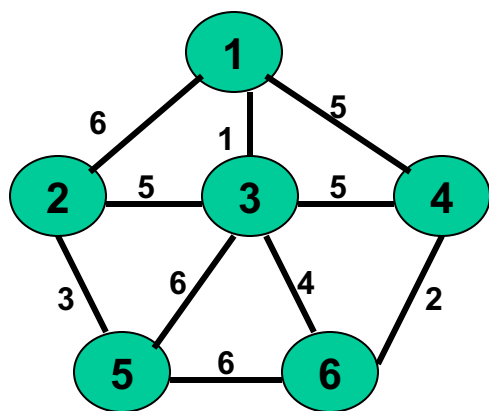
最小代价生成树
的生成过程

Prim 算法的实例



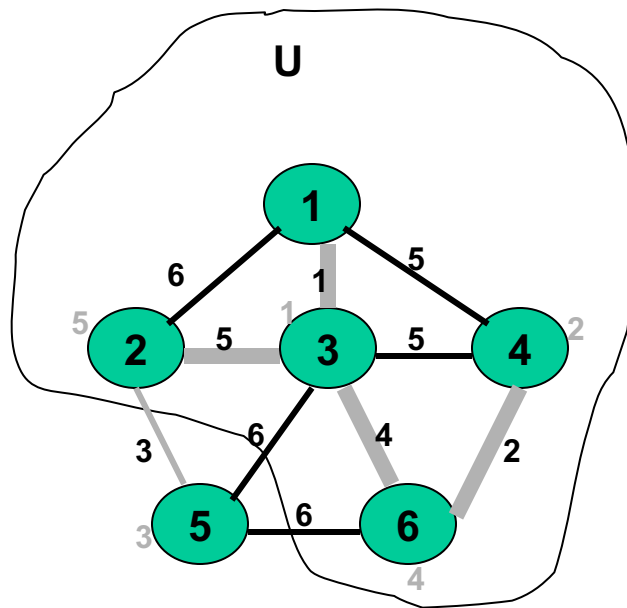
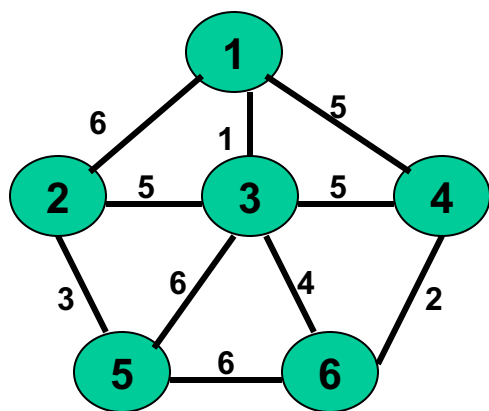
最小代价生成树
的生成过程

Prim 算法的实例



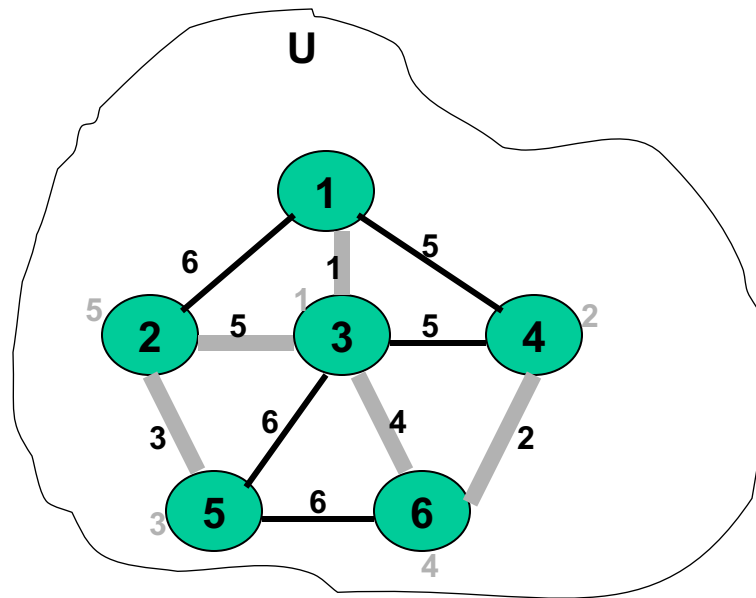
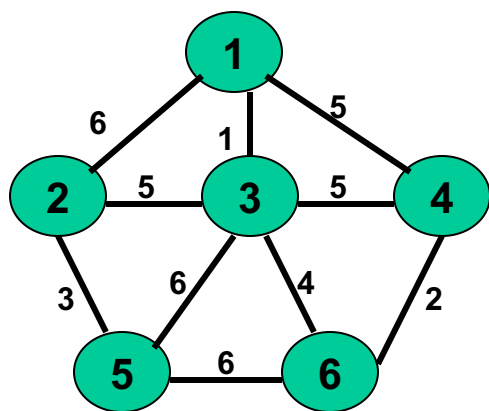
最小代价生成树
的生成过程

Prim 算法的实例



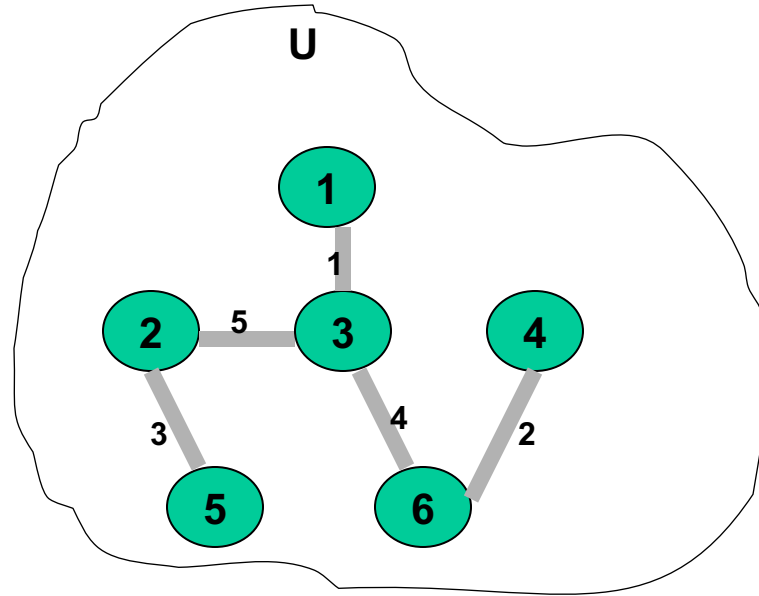
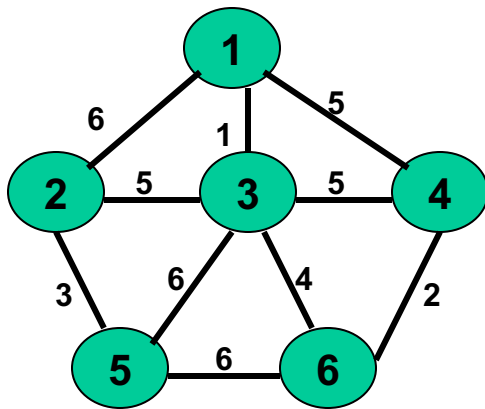
最小代价生成树
的生成过程

Prim 算法的实例



最小代价生成树
的生成过程

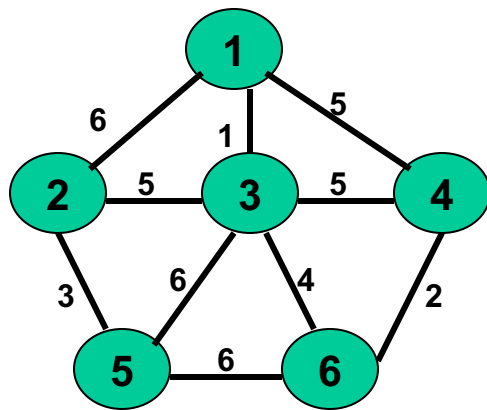
Prim 算法的实例



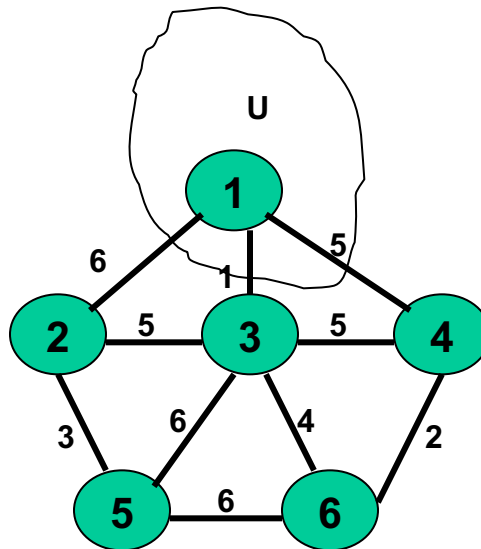
最小代价生成树
的生成过程

要点：每当新的结点并入 U 之后，则调整仍在 $V - U$ 集合中的结点至 U 中结点的最小距离。

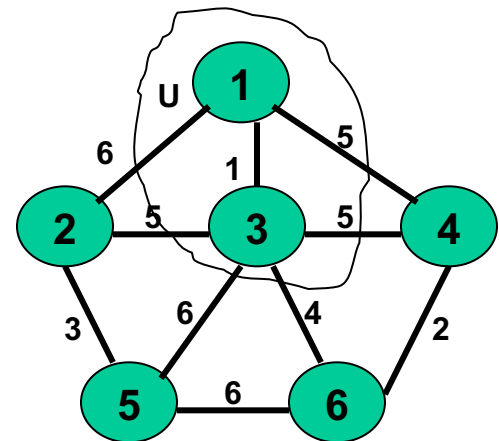
Prim 算法数据结构



图G



图G



图G

注意：第二步closedge[0].

Lowcost = 0

closedge[2]. Lowcost = 0 表示结

点 1 3 在U中。

lowcost 表示最小距离。

adjvex 表示相应结点(在V-U中)

。

数组：closedge[6]

0	0	
1	6	0
2	1	0
3	5	0
4	∞	0
5	∞	0

lowcost adjvex

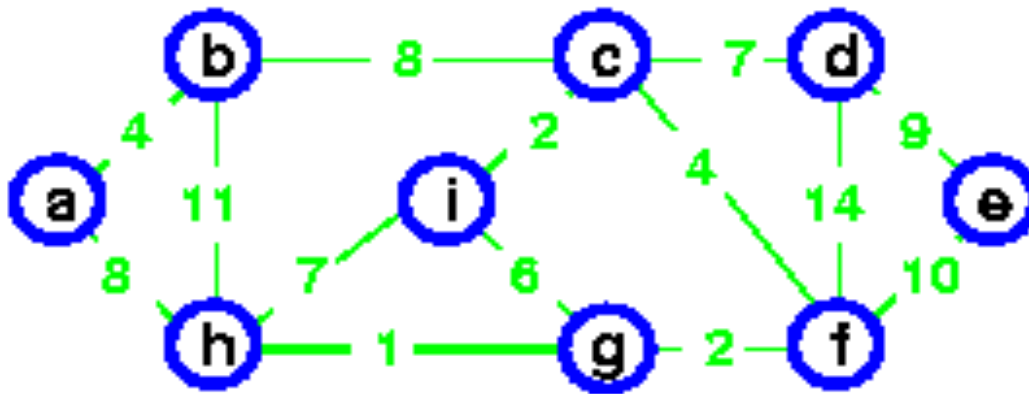
数组：closedge[6]

0	0	
1	5	2
2	0	
3	5	0
4	6	2
5	4	2

lowcost adjvex

Kruskal's Algorithm in operation

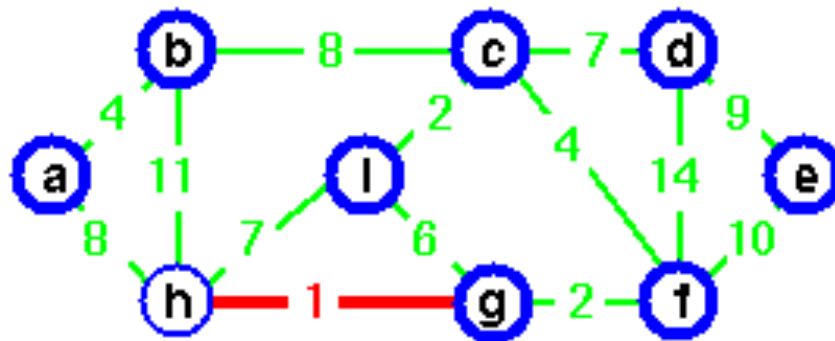
All the vertices are in single element trees



Each vertex is its own representative

Kruskal's Algorithm in operation

All the vertices are in single element trees

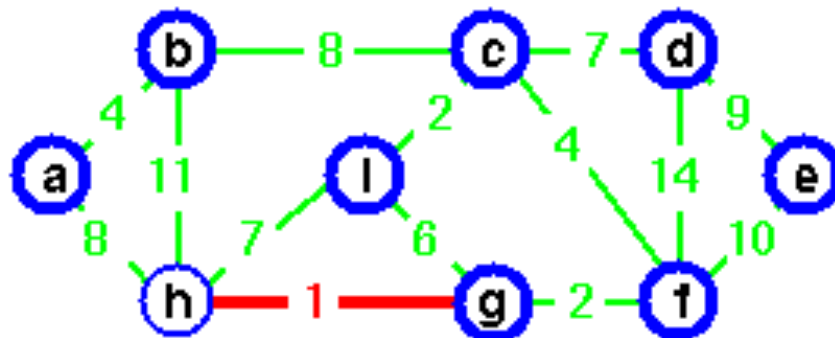


The cheapest edge
is h-g

Add it to the forest,
joining h and g into a
2-element tree

Kruskal's Algorithm in operation

The cheapest edge
is h-g

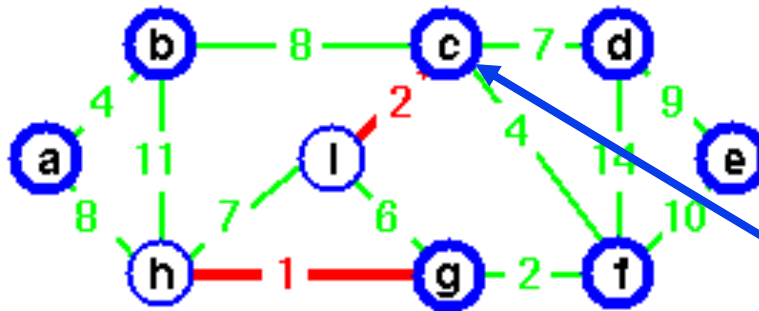


Add it to the forest,
joining h and g into a
2-element tree

Choose g as its
representative

Kruskal's Algorithm in operation

The next cheapest edge
is C-i



Add it to the forest,
joining C and i into a
2-element tree

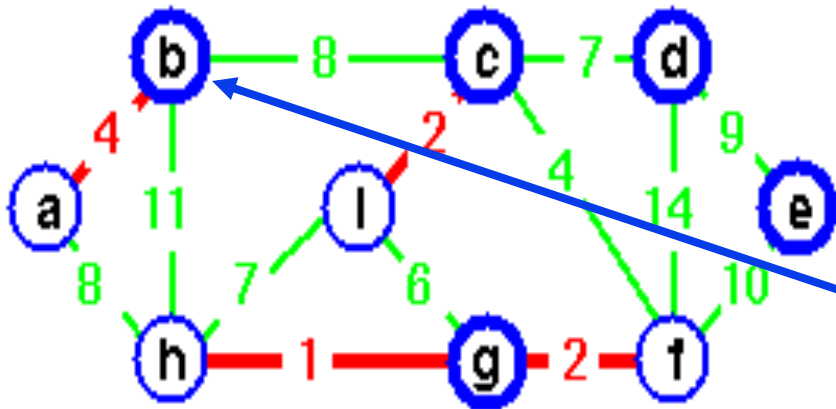
Choose C as its
representative

Our forest now has 2 two-element trees
and 5 single vertex ones

Kruskal's Algorithm in operation

The next cheapest edge
is a-b

Add it to the forest,
joining a and b into a
2-element tree



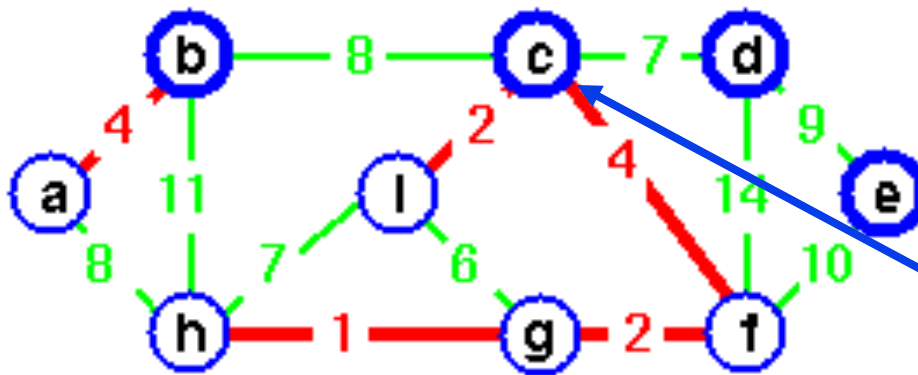
Choose b as its
representative

Our forest now has 3 two-element trees
and 4 single vertex ones

Kruskal's Algorithm in operation

The next cheapest edge
is **c-f**

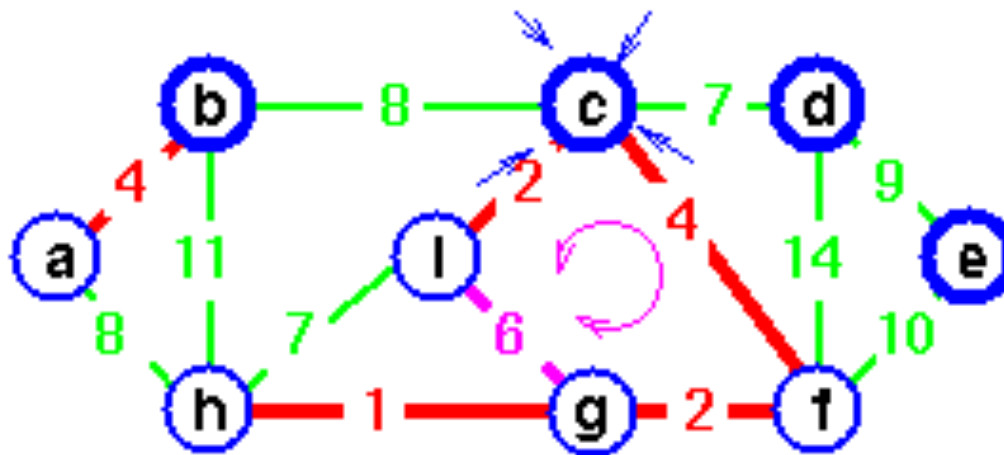
Add it to the forest,
merging two
2-element trees



Choose the rep of one
as its representative

Kruskal's Algorithm in operation

The next cheapest edge
is g-i



The rep of g is c

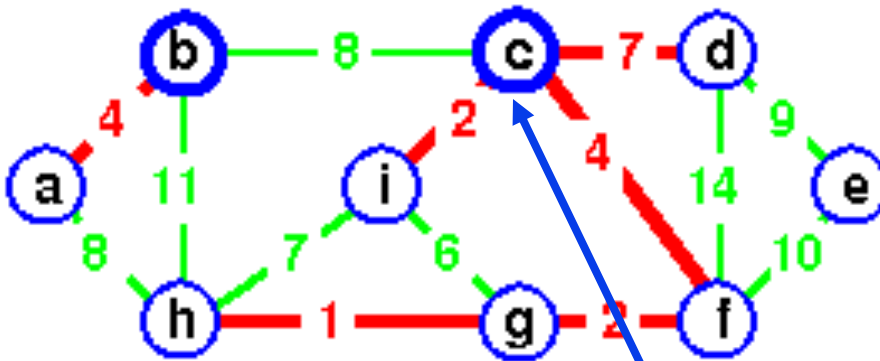
The rep of i is also c

\therefore g-i forms a cycle

It's clearly not needed!

Kruskal's Algorithm in operation

The next cheapest edge
is c-d



The rep of C is C

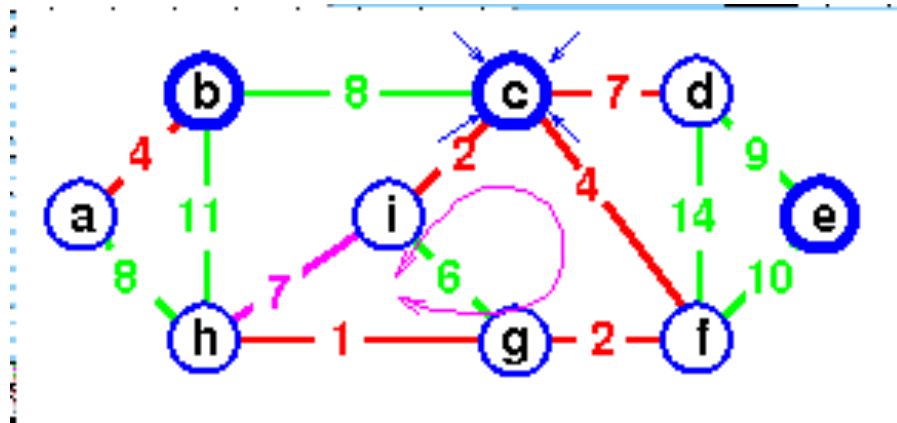
The rep of d is d

\therefore c-d joins two
trees, so we add it

.. and keep C as the representative

Kruskal's Algorithm in operation

The next cheapest edge
is h-i



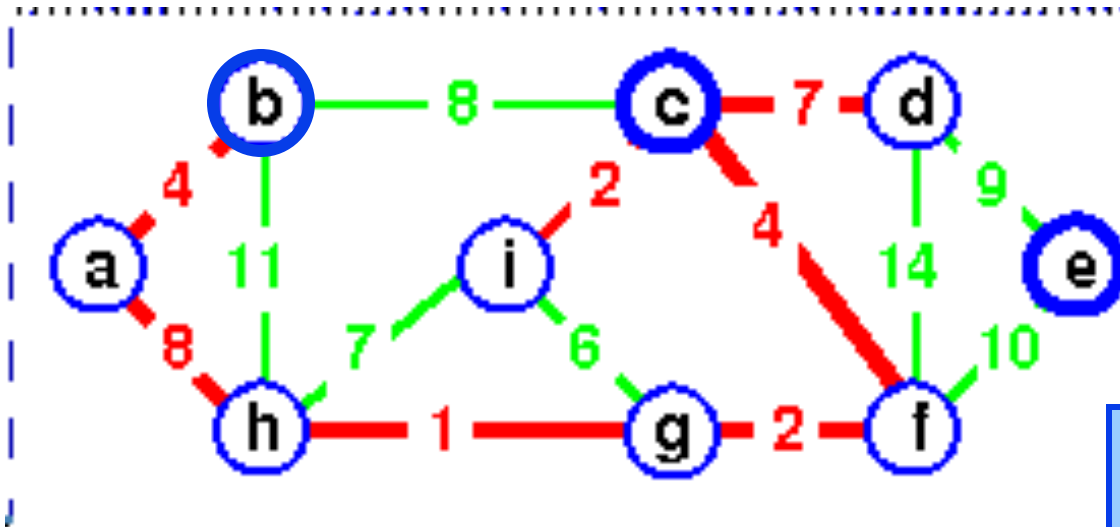
The rep of h is c

The rep of i is c

\therefore h-i forms a cycle,
so we skip it

Kruskal's Algorithm in operation

The next cheapest edge
is a-h



The rep of a is b

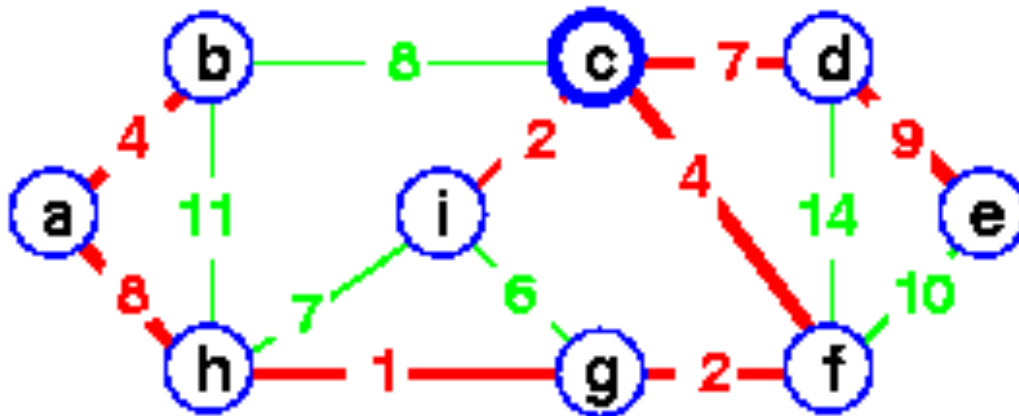
The rep of h is c

\therefore a-h joins two trees,
and we add it

Kruskal's Algorithm in operation

The next cheapest edge
is b-c

But b-c forms a cycle



So add d-e instead

... and we now have a spanning tree

Kruskal's algorithm

Algorithm 8.3 KRUSKAL

Input: A weighted connected undirected graph $G=(V,E)$ with n vertices

Output: The set of edges T of a minimum cost spanning tree for G

1. Sort the edges in E by nondecreasing weight
2. for each vertex $v \in V$
3. MAKESET($\{v\}$)
4. end for
5. $T = \{\}$
6. while $|T| < n-1$
7. Let (x,y) be the next edge in E
8. if $\text{FIND}(x) \neq \text{FIND}(y)$ then
9. Add (x,y) to T
10. UNION(x,y)
11. end if
12. end while

Prim's algorithm

Algorithm 8.3 PRIM

Input: A weighted connected undirected graph $G=(V,E)$, where $V=\{1,2,\dots,n\}$

Output: The set of edges T of a minimum cost spanning tree for G

1. $T \leftarrow \{\}$; $X \leftarrow \{1\}$; $Y \leftarrow V - \{1\}$
2. for $y \leftarrow 2$ to n
3. if y adjacent to 1 then $N[y] \leftarrow 1$, $C[y] \leftarrow c[1,y]$
4. else $C[y] \leftarrow \infty$
5. end for
6. for $j \leftarrow 1$ to $n-1$ {find $n-1$ edges}
7. Let $y \in Y$ be such that $C[y]$ is minimum
8. $T \leftarrow T \cup \{(y, N[y])\}$ {add edge $(y, N[y])$ to T }
9. $X \leftarrow X \cup \{y\}$, $Y \leftarrow Y - \{y\}$ {move vertex y from Y to X }
10. for each vertex $w \in Y$ that is adjacent to y
11. if $c[y,w] < C[w]$ then $N[w] \leftarrow y$, $C[w] \leftarrow c[y,w]$
12. end for
13. end for

File compression

Suppose we are given a file, which is a string of characters.

Compress the file as much as possible in such a way that the original file can be reconstructed.

Let the set of characters in the file be $C = \{c_1, c_2, \dots, c_n\}$. Let also $f(c_i)$, $1 \leq i \leq n$, be the frequency of character c_i in the file.

Since the frequency of some characters may be much larger than others, it is reasonable to use variable length encodings. When the encodings vary in length, we stipulate that the encoding of one character must not be the prefix of the encoding of another character; such codes are called prefix codes.

Huffman code

Algorithm 8.6 HUFFMAN

Input: A set $C=\{c_1, \dots, c_n\}$ of n characters and their frequencies $\{f(c_1), \dots, f(c_n)\}$

Output: A Huffman tree (V, T) for C

1. Insert all characters into a min-heap H according to their frequencies
2. $V \leftarrow C$; $T = \{\}$
3. for $j \leftarrow 1$ to $n-1$
4. $c \leftarrow \text{DELETEMIN}(H)$
5. $c' \leftarrow \text{DELETEMIN}(H)$
6. $f(v) \leftarrow f(c) + f(c')$ $\{v \text{ is a new node}\}$
7. $\text{INSERT}(H, v)$
8. $V = V \cup \{v\}$ $\{\text{Add } v \text{ to } V\}$
9. $T = T \cup \{(v, c), (v, c')\}$ $\{\text{Make } c \text{ and } c' \text{ children of } v \text{ in } T\}$
10. end while

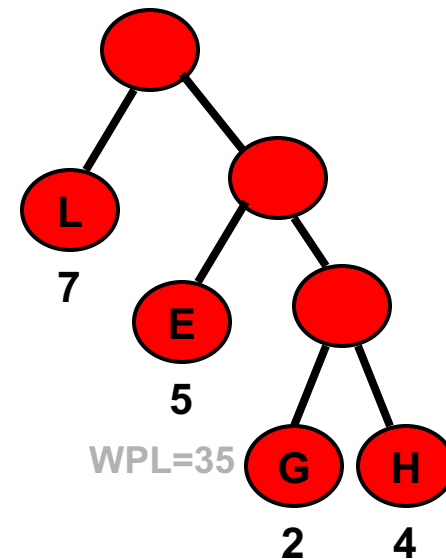
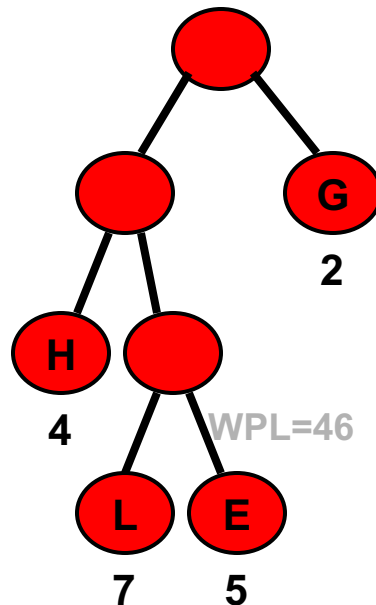
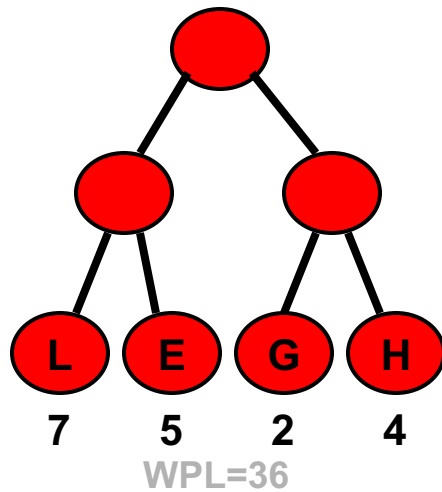
Time complexity: $O(n \log n)$

1、最优二叉树（赫夫曼树）

- 路径长度：结点之间的树枝的总数
- 树的路径长度：从根到每一结点的路径长度之和
- 树的带权路径长度：叶子结点的带权路径长度之和。设有 n 片叶子，它们的权值分别为 w_1 、 w_2 、..... w_n ，相应的路径长度分别为 L_1 、 L_2 、..... L_n 。

则树的带权路径长度可记为：

$$WPL = \sum_{k=1}^n w_k l_k$$



- 最优二叉树或赫夫曼树：树的带权路径长度 **WPL** 最小的二叉树。

赫夫曼算法

1、最优二叉树（赫夫曼树）

• 赫夫曼算法（产生最优二叉树的算法）的实现：

1、给定一个具有 n 个权值 $\{w_1, w_2, \dots, w_n\}$ 的结点的集合

$F = \{T_1, T_2, \dots, T_n\}$ 。

2、初始时，设 $A = F$ 。

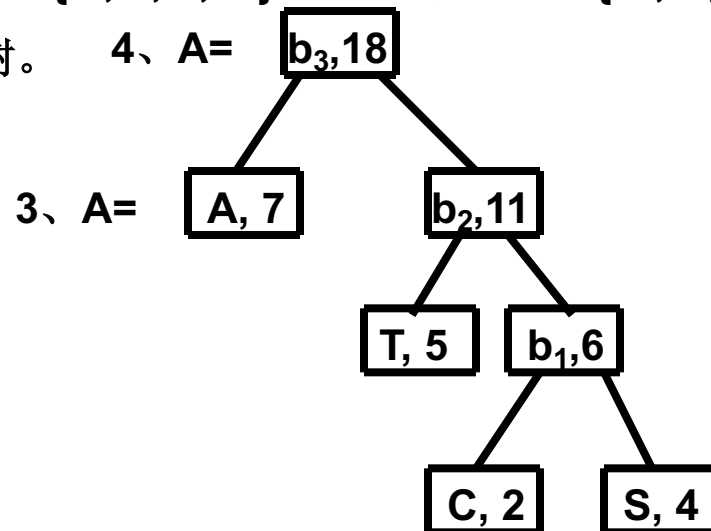
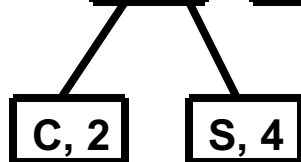
3、执行 $i = 1$ 至 $n-1$ 次循环，在每次循环时，做以下事情：

从当前集合中选取权值最小、次最小的两个结点，以这两个结点作为内部结点 b_i 的左右儿子， b_i 的权值为其左右儿子权值之和。在集合中删除这两个权值最小、次最小的结点。这样，在集合 A 中，结点个数便减少了一个。

e.g: 已知权值（此处为使用概率）分别为 $\{2, 7, 4, 5\}$ 的结点集合 $F = \{C, A, S, T\}$ ，
请利用赫夫曼算法产生最优二叉树。

1、 $A =$ C, 2 A, 7 S, 4 T, 5

2、 $A =$ $b_1, 6$ A, 7 T, 5

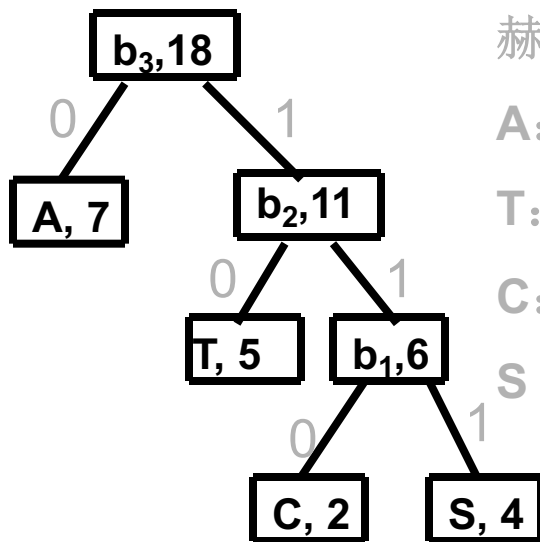


赫夫曼编码

2、赫夫曼编码

- 赫夫曼算法用于通信中的字符的编码。将权值当成使用概率。赫夫曼树的左分支 标记为 0，而右分枝标记为 1；这从根到每一个叶子结点（字符）的路径上标记的字符组成的字符串，即为该字符的赫夫曼编码。

e.g:已知权值（此处为使用概率）分别为 $\{2, 7, 4, 5\}$ 的结点集合 $F = \{C, A, S, T\}$ ，请利用赫夫曼算法产生赫夫曼编码。



赫夫曼编码：

A: 0
T: 10
C: 110
S: 111

赫夫曼编码优点：

占用二进制位少

e.g: 左图发送长度为 n 的字符串，等长表示需 $2n$ 个比特。因共有四个字符，表示每个字符需二个 比特。

采用赫夫曼编码后，总的比特数 $35n/18$ ，因：

A: $1 \cdot 7n / 18$

T: $2 \cdot 5n / 18$

S: $3 \cdot 4n / 18$

C: $3 \cdot 2n / 18$

赫夫曼编码

2、赫夫曼编码

• 赫夫曼算法的实现：

1、建立具有 $2n-1$ 个单元的数组，其中 n 个单元用于保存初始结点， $n-1$ 个结点用于表示内部结点。

2、执行 $n-1$ 次循环，每次产生一个内部结点。权值最小的两个结点为其左右儿子。

3、计算每个字符的赫夫曼编码。

• 数据结构：

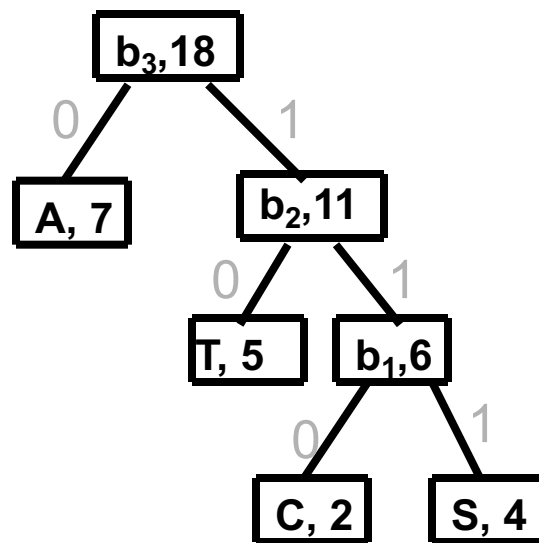
```
typedef struct
```

```
{ unsigned int weight ;
```

```
    unsigned int parent, lchild, rchild ;
```

```
} HTNode, * HuffmanTree;
```

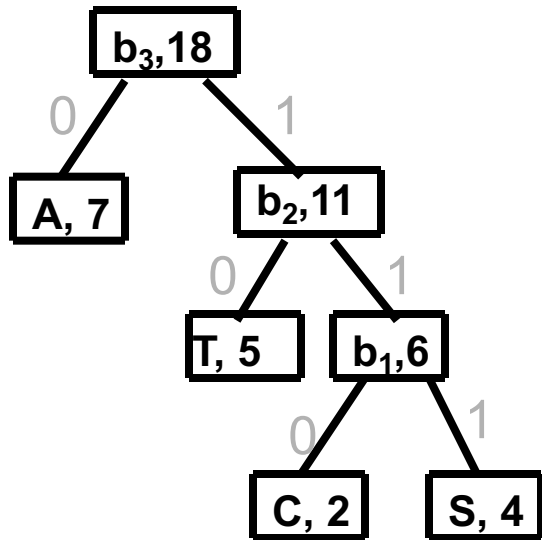
```
typedef char * * HuffmanCode ;
```



赫夫曼编码

2、赫夫曼编码

• 赫夫曼算法的程序实现


$$0 \ 1 \ \dots \ n \ n+1 \ n+2 \ \dots \ 2n-1$$

□ □ □ ○ ○ ○

注意: **Select** 子程序不考虑父结点地址不为 0 的结点,范围: 1~i-1

$$0 \quad 1 \quad 2 \quad \dots \quad n-2 \quad n-1$$

--	--	--	--	--	--

cd 字符数组

```

Void HuffmanCode ( HuffmanTree &HT, HuffmanCode &HC, int *w, int n )
if ( n <= 1 ) return; m = 2 * n -1;
HT = ( HuffmanTree ) malloc (( m + 1 ) * sizeof ( HTNode )); // 不用 0 号单元
for ( p = HT, i = 1, p++; i <= n; ++i, ++p, ++w ) *p = { *w, 0, 0, 0 };
for ( ; i <= m; ++i, ++p ) *p = { 0, 0, 0, 0 };
for ( i = n +1; i <= m; ++i )
{ Select ( HT, i -1, s1, s2 ); // 可以用堆或优先队列，使时间降低到 O (logn)
  HT[s1].parent = i; HT[s2].parent = i; HT[i].lchild = s1; HT[i].rchild = s2 ;
  HT[i].Weight = HT[s1].Weight + HT[s2].Weight;
}
HC = ( HuffmanCode ) malloc (( n + 1 ) * sizeof ( char * ));
Cd = ( char * ) malloc ( n * sizeof ( char ) ); cd[ n -1 ] = “ /0” ;
for ( i = 1; i <= n; ++i )
{ start = n-1;
  for ( c = i, f = HT[ i ].parent; f != 0; c = f, f = HT[ f ].parent )
    if ( HT[ f ].lchild == c ) cd[ - - start ] = “ 0” ; else cd [ - - start ] = “ 1” ;
    HC[i] = ( char * ) malloc (( n - start ) * sizeof ( char * ));
    strcpy(HC[i], &cd[start]);
  }
free(cd);
}

```

- 最短路径、最小生成树、Huffman编码算法正确性证明