# Summary of Algorithms

- Scheduling
  - Integer/fraction Knapsack   CLRS 16.2
  - Job Schedule   Ex. 16-2 (a)
  - Deadline Schedule   CLRS 16.5

- KMP String matching   CLRS 32

- Union/Find Algorithm   CLRS 21
  - Union by height/size
  - Path compression
    - $O(M \log^*(N))$ explained

- **Capacity graph:** link i ! j has capacity c(i,j)

- <u>Max Flow Problem</u>: Max flow into source node (s) out of terminal node (t)

  - ◆ constraints on flow i to j: f(i,j) = - f(j,i) $\sum_{j} f(i,j) = 0$

    capacity: 0 · f(i,j) · c(i,j) ;
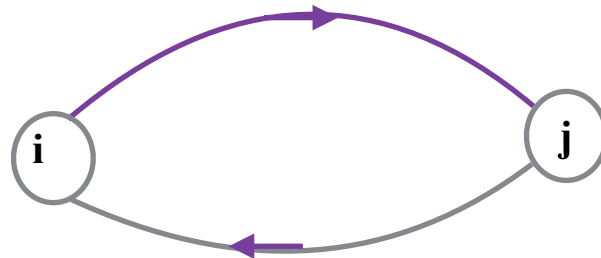
If there is return arc   -c(j,i) · f(i,j) ·   c(i,j)

Augmentation, Flow, Residual

- Initialize residual graph G(N,A1), A1 = A.

- Initialize all f(i,j) = 0. $0 \cdot \leq r(i,j) \leq+ c(i,j)$

    a. Find path P from s to t (**augmenting path**)

    b. Find minimum capacity arc on path P. Denote capacity by C.

    c. Add C units of flow f(i,j) -> f(i,j) + C to each arc on path Modify residual graph (N,A1) accordingly.

    $$- f(i,j) < r(i,j) \cdot < c(i,j) - f(i,j)$$

    d. Repeat a-c until no path P from s to t in G(N,A1)

- Properties:

    a) Total flow increases and terminate when no augmentation possible: $O(Max(c) |A|)$

    b) With minimum hop path for augmentation $O(|N| |A|^2)$
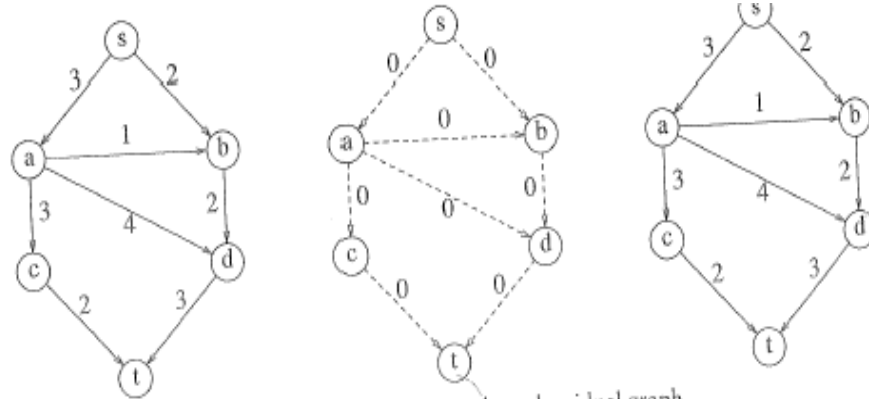
Solution without backtracking

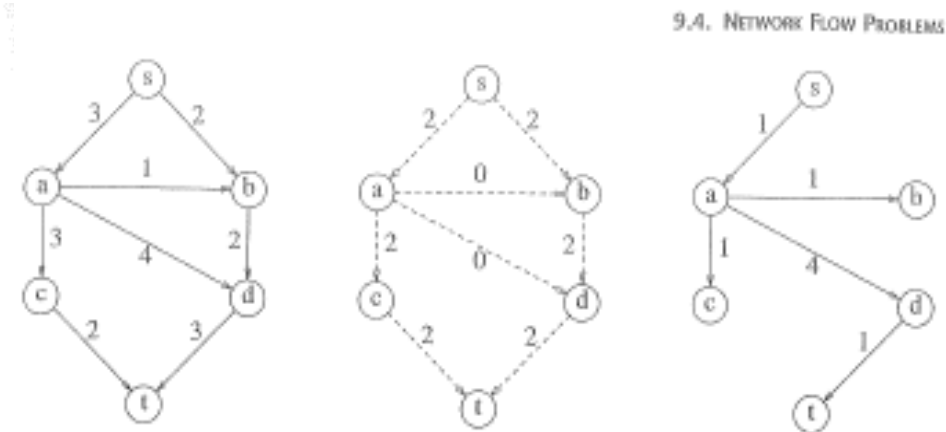Figure 9.40 Initial stages of the graph, flow graph, and residual graph

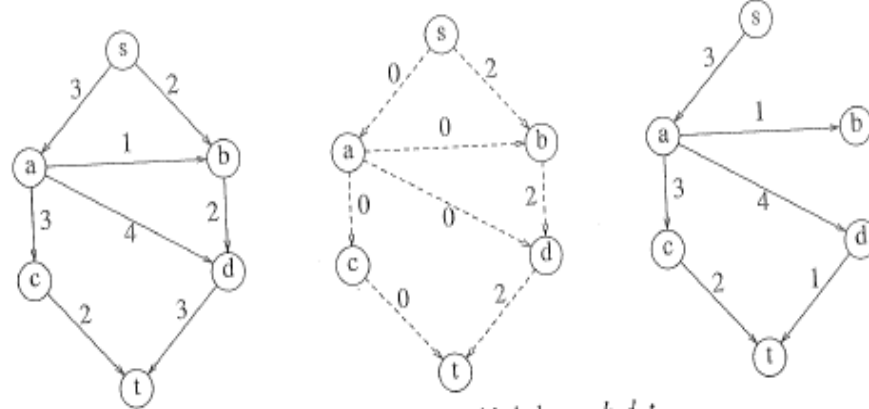Figure 9.41 G, G_f, G_r after two units of flow added along s, b, d, t

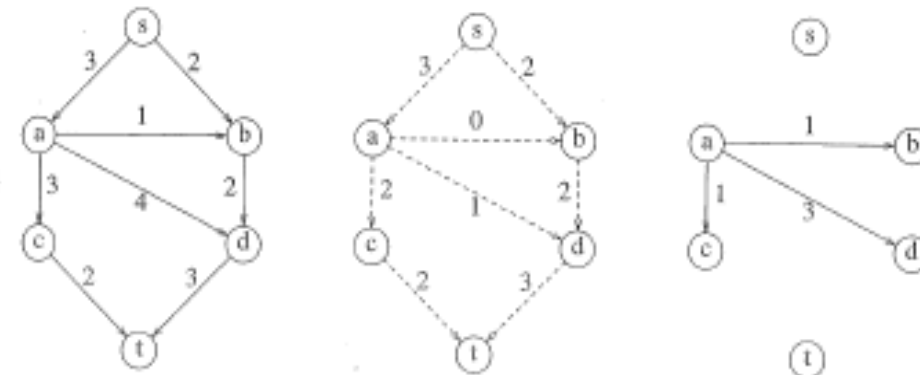Figure 9.42 G, G_f, G_r after two units of flow added along s, a, c, t

Figure 9.43 G, G_f, G_r after one unit of flow added along s, a, d, t—algorithm terminates
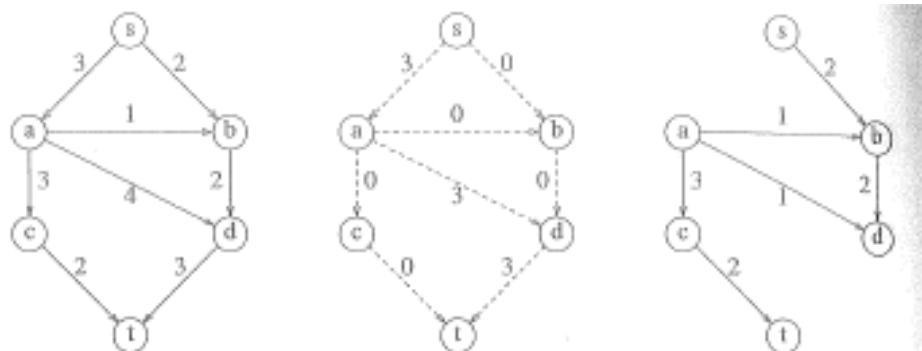
Figure 9.44 G, G_f, G, if initial action is to add three units of flow along s, a, d, t—algorithm terminates with suboptimal solution
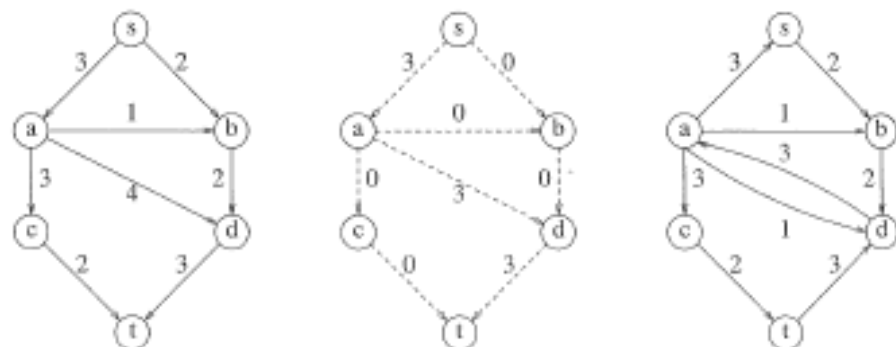

Figure 9.45 Graphs after three units of flow added along s, a, d, t using correct algorithm
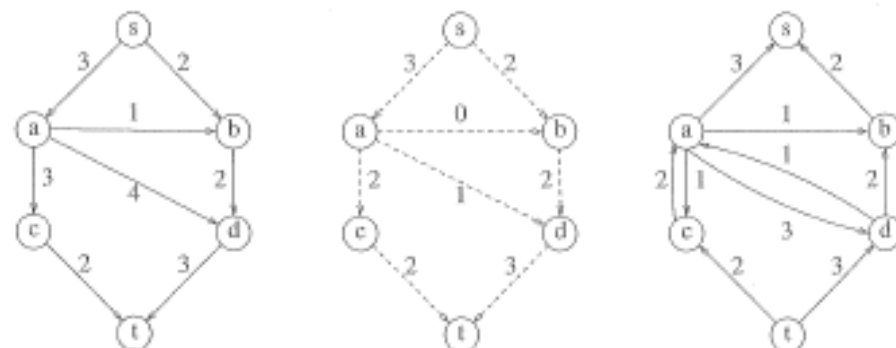

Figure 9.46 Graphs after two units of flow added along s, b, d, a, c, t using correct algorithm
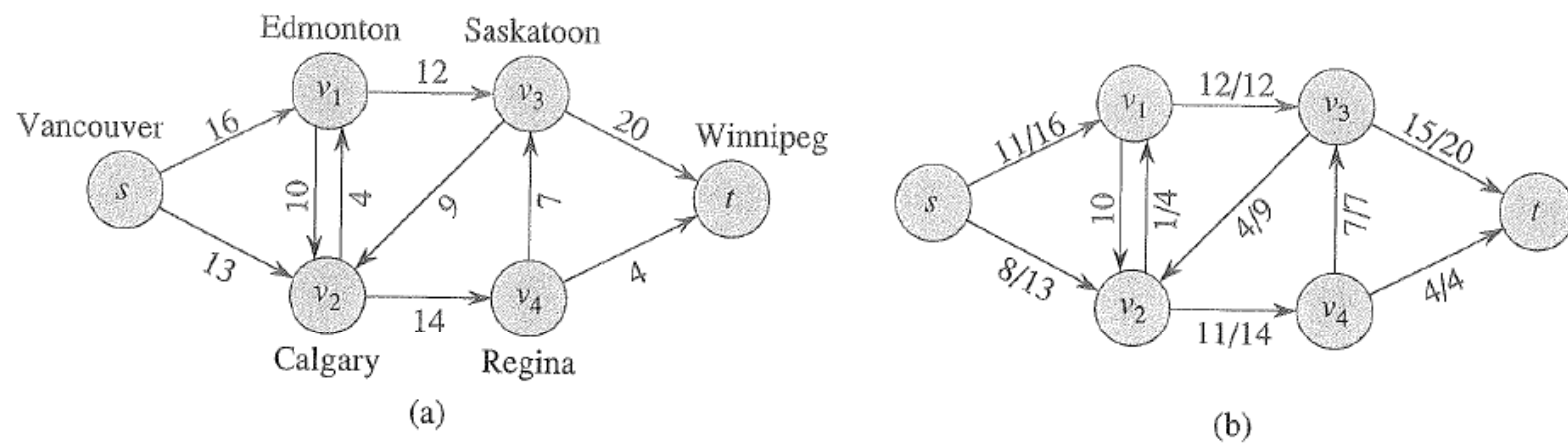
Solution with backtracking

**Figure 26.1** (a) A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. Pucks are shipped through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v. Each edge is labeled with its capacity. (b) A flow f in G with value $|f| = 19$. Only positive flows are shown. If $f(u, v) > 0$, edge (u, v) is labeled by $f(u, v)/c(u, v)$. (The slash notation is used merely to separate the flow and capacity; it does not indicate division.) If $f(u, v) \leq 0$, edge (u, v) is labeled only by its capacity.

Residual Graph   *Flow/Capacity Graph*
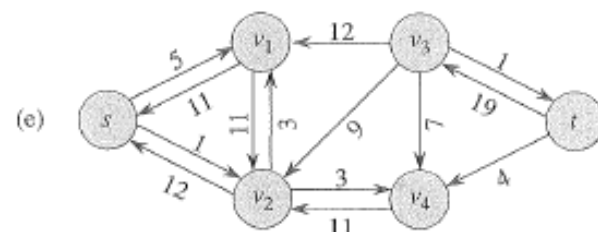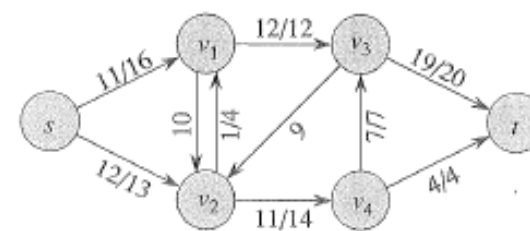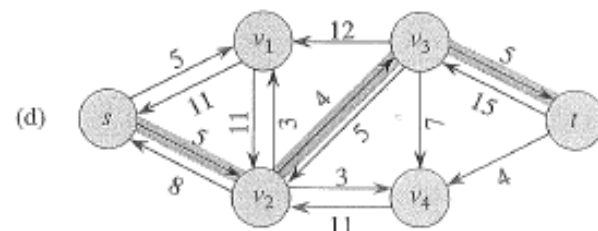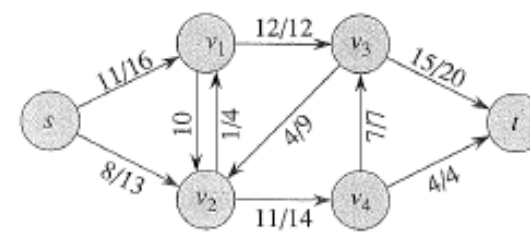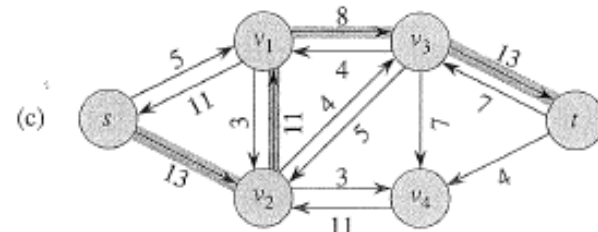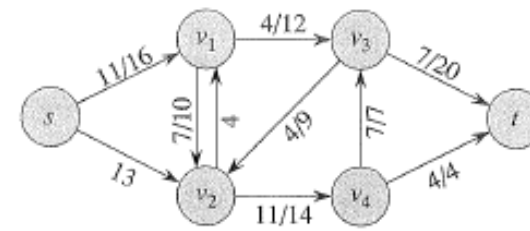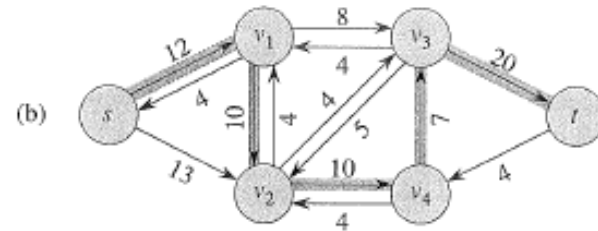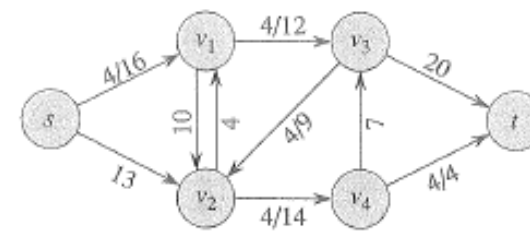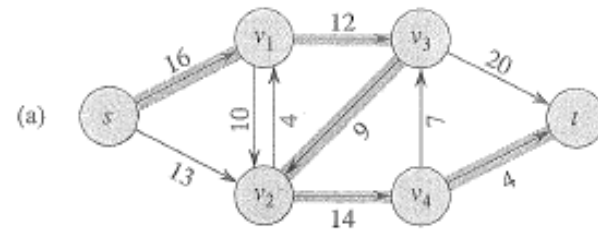
Classical Bad Case
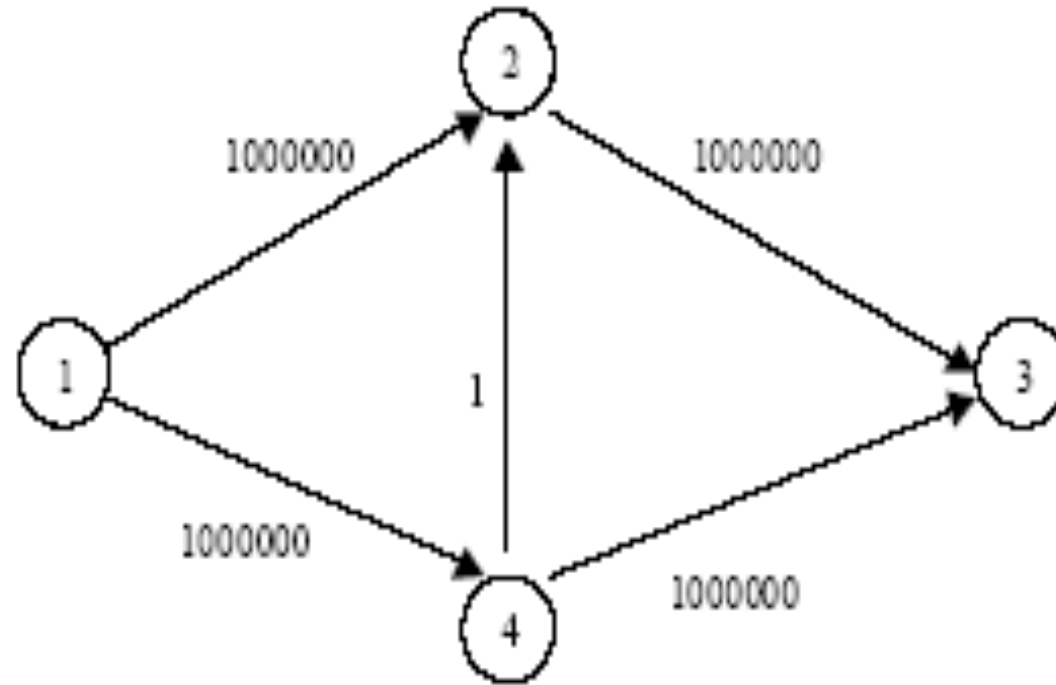
Scheduling & Linear Programming

- *Knapsack*
- *Queuing*
- *Schedule with Deadlines*

Fractional Knapsack

- *Given set of task that take "time"* $t_1, t_2, t_3, …, t_N$ *and value* $v_1, v_2, v_3, …, v_N$

Allocate to maximize objective

$$S(x_i) = \Sigma_i \, x_i \, V_i$$

with limited resource T:

$$\Sigma_i \, x\_i \, t_i \, <= T$$

Problem is to find assignment fractions

$$0 < x_i < 1$$

(Put most valuable parts of N objects in to your sack)

Solution:

- Sort in increasing value per time v_i/t_i and take them in order as far as possible:

- All: $x_1 = 1$, $x_2 = 1$, ..., $x_{m-1} = 1$, Some of $0 \cdot x_m \cdot 1$, None: $x_{i+m} = 0$, ..., $x_N = 0$

- So that $\quad t_1 + t_2 + ... t_{m-1} + x_m t_m = T$ exactly $\quad 0 \cdot x_m = (T - t_1 - t_2 - ... - t_{m-1})/t_m \cdot 1$

Maximize Rate of Return

Integer Knapsack: x_i = 0 or 1 is VERY HARD to SOLVE!

Example: T = 50

Objects: v/t = $60 / 10, $100/20, $120/30

Greedy $60 + $100 = $160
Try it: Best integer is $100 + $120 = $220
Best fractional: $240

See CRLS Fig. 16.2
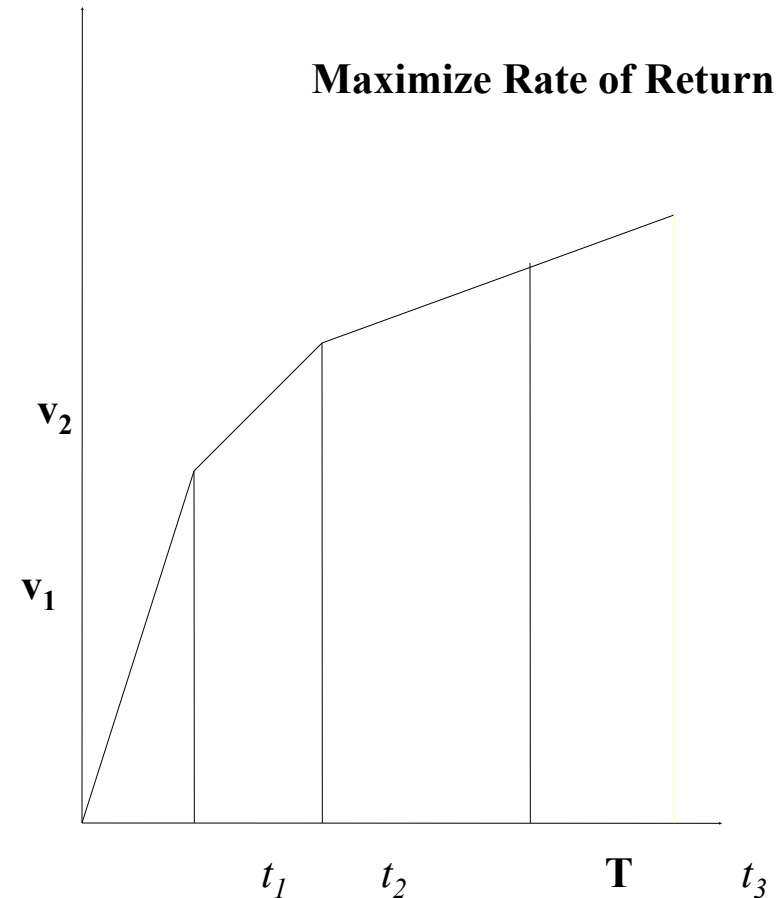
Queuing: Minimize "time in line"

- Jobs in queue take time t_i

- Minimize total waiting time:

$t_1 + (t_1 + t_2) + (t_1+t_2+t_3) + \ldots + (t_1 + \ldots + t_N)$

$$W = N\, t_1 + (N-1)\, t_2 + \boxed{?} + t_N$$

- Solution: Min W by sorting t's in ascending order (shorter jobs first)!

- Recall sorting is by "swap theorem"

$$\text{Sort} \equiv \text{Max over permutation} \sum_i i\, a[i]$$

Scheduling with deadlines

- Task all take same time: $\triangle$ T but they have different values and deadlines

- Sort in descending values (or penalties!):

  v_1 ، v_2 ، .. ، v_N

  d_1, d_2,   , d_N

- Feasible Solutions can always be in DFF!

- So this is the contraction:

  - Select them one at a time from ordered list of v's

  - Schedule them in order in "early first form" (EFF).

  - If schedule fails drop last one selected and continue to end of list.

  Alternative solution: maximum procrastination!

CRLS Fig 16.7

a	1	2	3	4	5	6	7
d	4	2	4	3	1	4	6
v	70	60	50	40	30	20	10

a4   a2   a3   a1   a7

Select a1, a2,a3 and a4

Reject a5 and a6

Select a7

Knuth-Morris-Pratt string matching
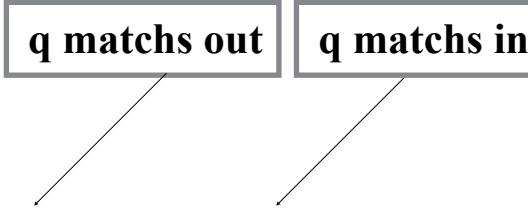
- Text: T(1),T(2),...,T(N)
- Pattern: P(1), P(2),...,P(M)
- Match if
  - ◆ T(s+1) =P(1), T(s+2) = P(2),.., T(M) = P(M)
  - ◆ or T(s+1:s+M) = P(1:M)
- Trivial scan O(M (N-M+1))
- KMP algorithm O(N+M) by amortized analysis

Prefix function:

- Given a partial match $P(1:q) = T(s+1:s+q)$ what is the smallest shift that matches end of $T(s+1:s+q)$

- It is $q - pi(q)$ where $pi(q)$ prefix function for the max match of prefix to suffix of $P(1:q)$

- $pi(q) = Max\{k<q$ s.t. $P(1:k) = P(1+q- k:q\}$

  i.e. q matches become pi(q) ds $= q - pi(q)$

- Strategy pre-compute $pi(q)$ and use it to advance the match.

KMP Algorithm

■ Compute pi(q) set q = 0

■ For i = 1, N

  { // count c_i at while $c_i + q(i) - q(i-1) < =1$

    while q>0 and P(q+1) \= T(i) set q =pi(q) ;

    if P(q+1) = T(i) then q = q+1 ;

    if q = M {

      report pattern found T(i – M;i);

      q = pi(q);

        }

  }

| q matchs out | q matchs in |

b a c b a b a b a a b c b a b T

a b a b a c a P

s

q

(a)

b a c b a b a b a a b c b a b T

$s' = s + 2$

a b a b a c a P

k

(b)

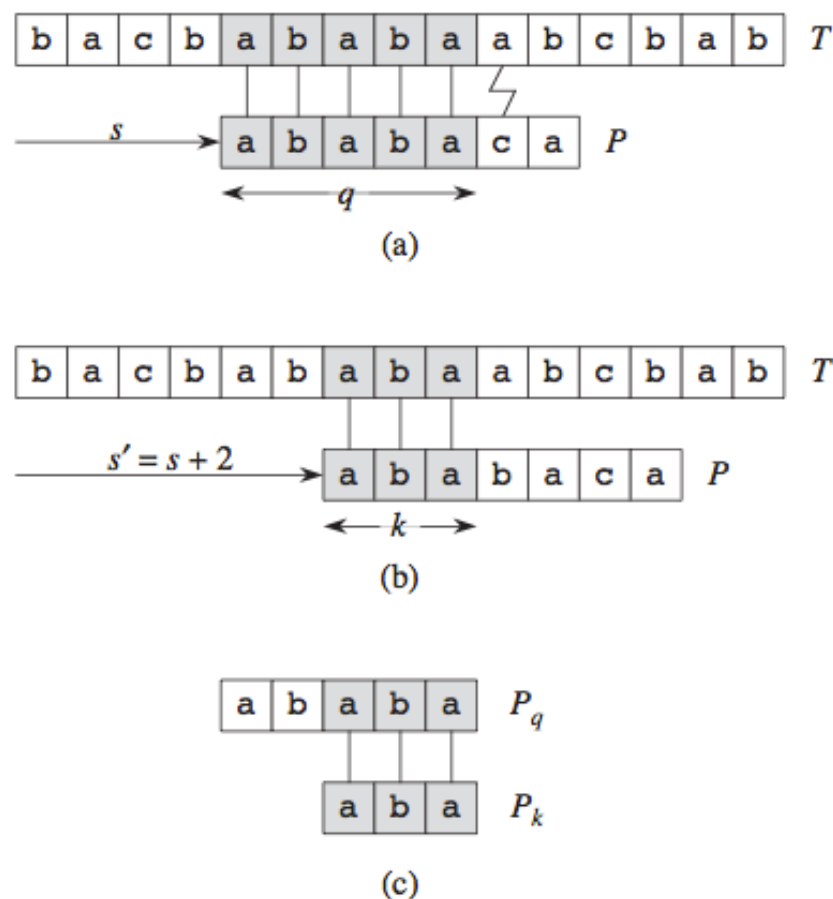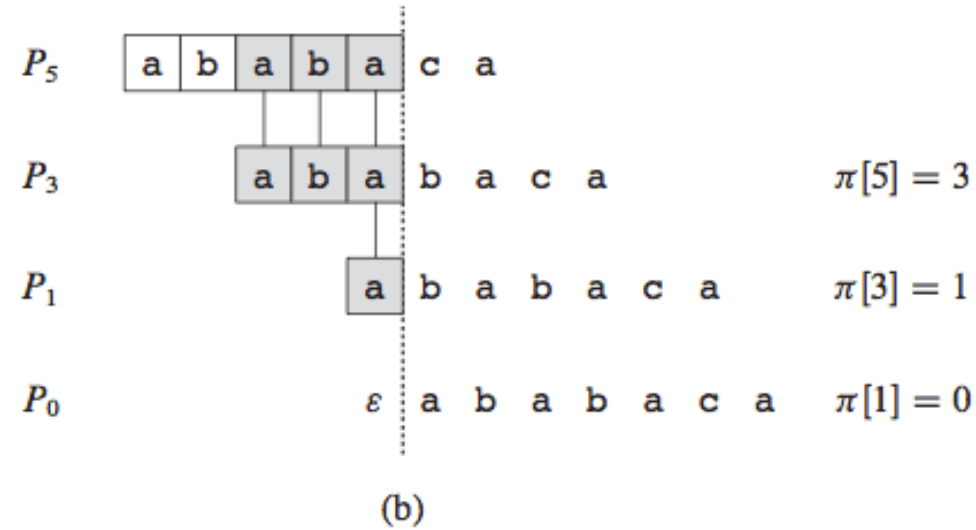a b a b a P_q

a b a P_k

(c)

Figure 32.10   The prefix function π. **(a)** The pattern $P =$ ababaca aligns with a text T so that the first $q = 5$ characters match. Matching characters, shown shaded, are connected by vertical lines. **(b)** Using only our knowledge of the 5 matched characters, we can deduce that a shift of $s + 1$ is invalid, but that a shift of $s' = s+2$ is consistent with everything we know about the text and therefore is potentially valid. **(c)** We can precompute useful information for such deductions by comparing the pattern with itself. Here, we see that the longest prefix of P that is also a proper suffix of P_5 is P_3. We represent this precomputed information in the array π, so that $\pi[5] = 3$. Given that q characters have matched successfully at shift s, the next potentially valid shift is at $s' = s+(q-\pi[q])$ as shown in part (b).

KMP Amortize analysis

- At While statement
- All sets are O(N) except at the while statement dq is called c(i) times
- $c(i) + q(i) - q(i-1) <= 1$ worst case.
- $sum\_i (c(i) + q(i) - q(i-1)) <= N$
- $sum\_i c(i) <= N + q(0) - q(N) <= N$
- Therefore is O(N)
- Construction of pi(M) is O(M)

(a)

(b)

- Set pi(1) = 0; q = 0
- For i= 2,.., M

Computing pi(q)

```
{
    while q>0 and P(q+1) \= P(i) set q =pi(q) ;
        if P(q+1) = P(i)  { q = q+1;   pi(i) = q; }
}
```

Relations: *Boolean valued Matrix   R[a,b]*

- Set: S = {a,b,c,....}

- Relation   (a,b) 2 S x S: a R b is True?

- Properties:

  - Reflexive: a R a is True
  - Anti-symmetric: a R b and b R a ➔ a = b
  - Transitive: a R b and b R c ➔ a R c
  - Total Ordering: a R b or b R a (inclusive or)
  - Self dual: a R b ⬅➔ b R a
  - Transpose: a R b ⬅➔ b R^T a

- RAT is partial ordering: e.g. descendants in a tree!

(e.g. <= is total ordering for int but g(N) = O(f(N)) is partial ordering!)

- Equivalence class is Reflexive, Transitive and Symmetric

  - Symmetric   a R b if and only if b R a

Union/Find

- Equivalence class and Sets.
- O(1) Find
- O(1) Union
  - Union by Height/Size
  - Path Compressions
  - $\text{Log}^*(N)$ function
- Binomial Queue

Dynamic Equivalence

- Object a[i] can be numbered 0,...,N-1 (like nodes)
- Sequence of new equivalence a ~ b
- Two operations:
- FIND a in Si ?
  - ◆ Must return T/F for find(a) ~find(b)
- UNION Sk = Si U Sj
  - ◆ Operation of find(a) not~ find(b)

- Want M finds and upto N unions: O(M+N)?
- Almost but actually impossible!

Determine if a ~ b

- O(1) answer is a ~`b if use array of Set #
- Set up a 2-d are for SxS and look up T/F
- Alternatively "partition" S into equivalence classes (like connected component) is C_i for all a's ~ b's
- S = C1 U C2 U ...U Cn and C's are disjiont: Ci ^ Ci = 0 (null set).
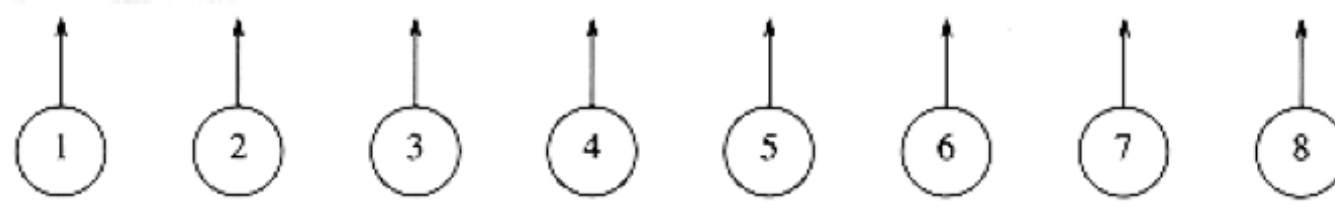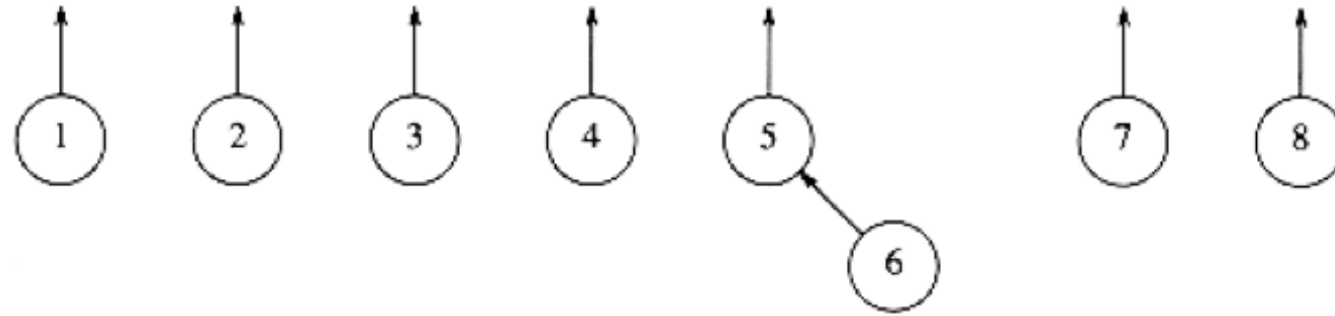
Figure 8.1 Eight elements, initially in different sets
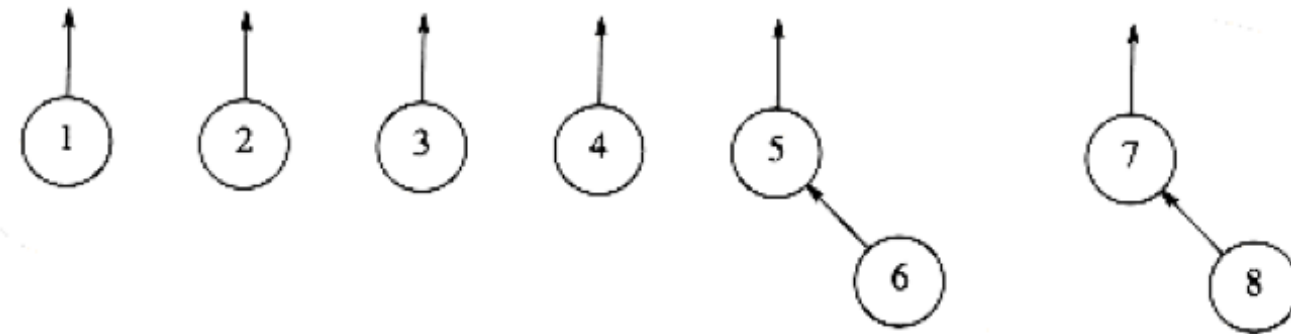

Figure 8.2 After union (5, 6)

Figure 8.3 After union (7, 8)

23

Figure 8.4 After union (5, 7)
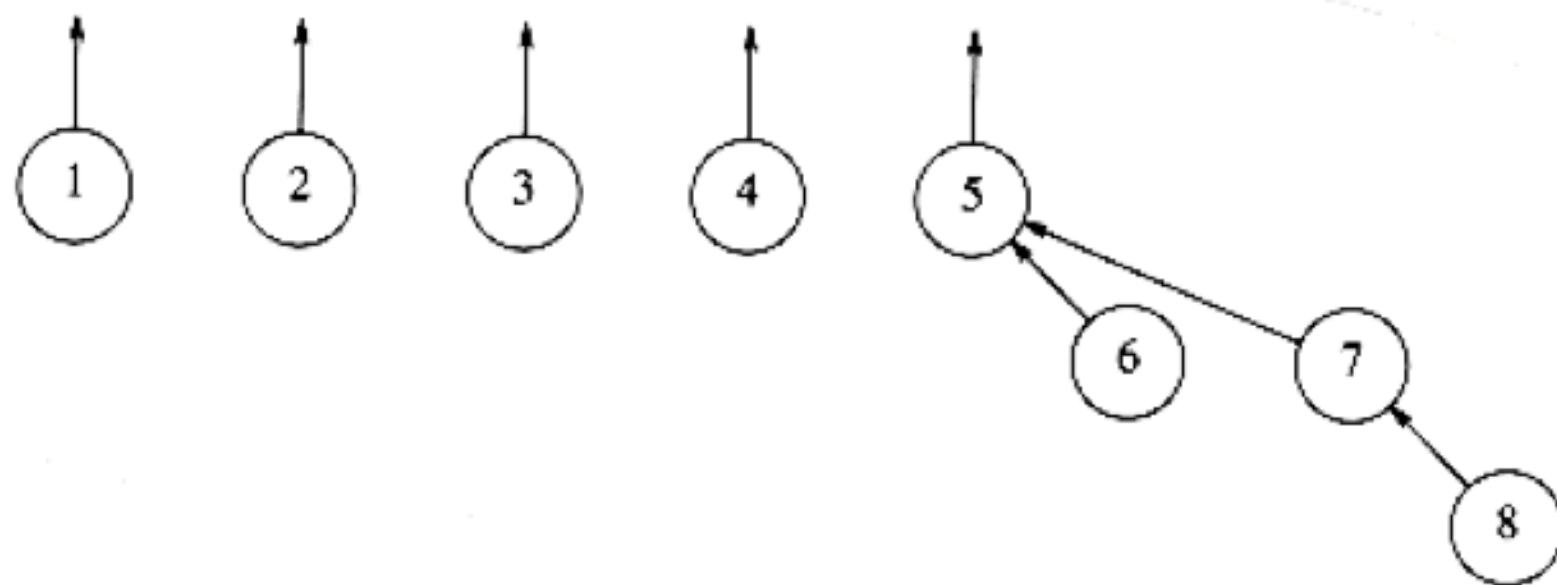
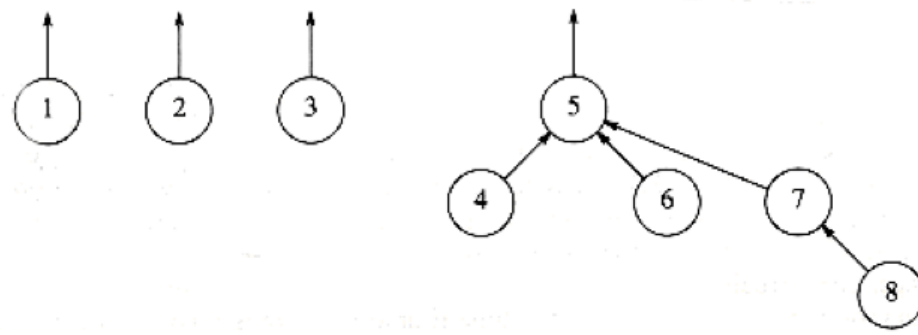| 0	0	0	0	0	5	5	7
1	2	3	4	5	6	7	8

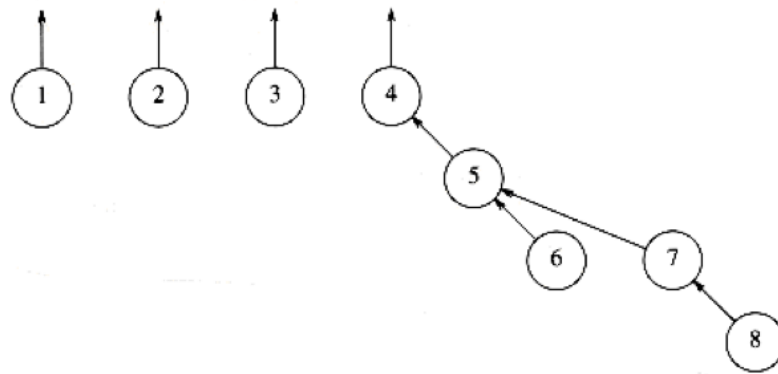Figure 8.10 Result of union-by-size

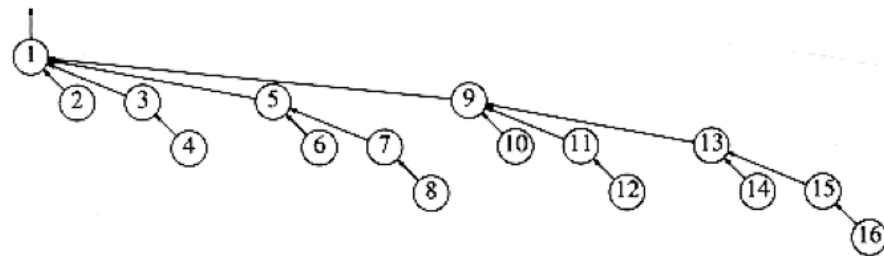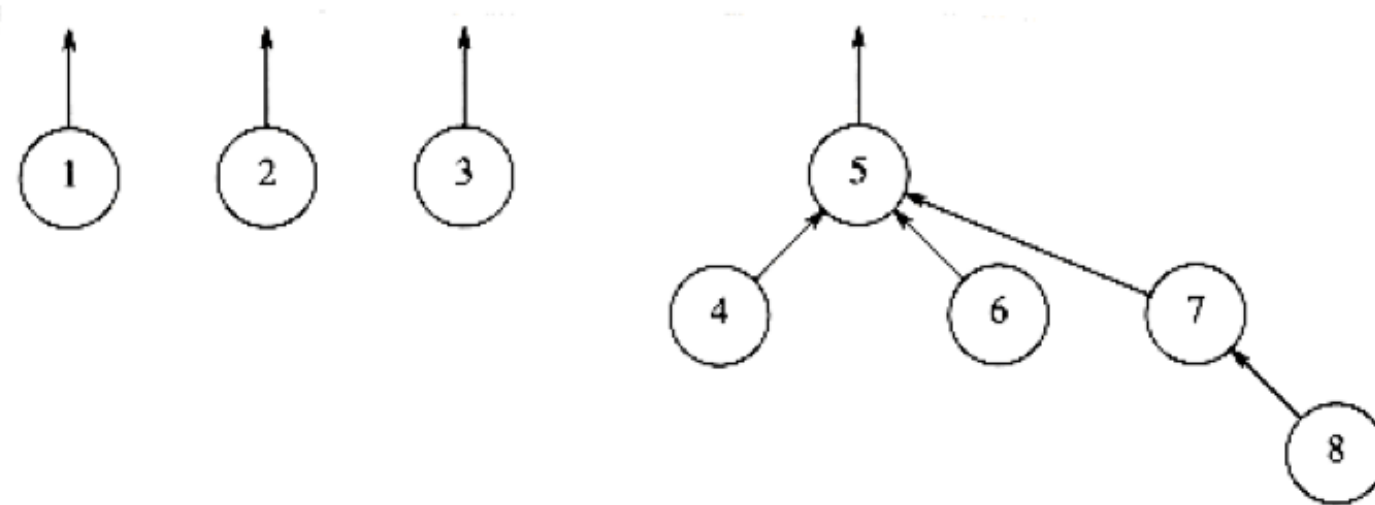Figure 8.11 Result of an arbitrary union

Figure 8.12 Worst-case tree for n = 16

23

The following figures show a tree and its implicit representation for both union-by-size and union-by-height. The code in Figure 8.13 implements union-by-height.

−1	−1	−1	5	−5	5	5	7	- size
1	2	3	4	5	6	7	8	

0	0	0	5	−2	5	5	7	- height
1	2	3	4	5	6	7	8	

23

Log*(N) inverse Ackermann function:

- *Ackerman Function:*
  - ◆ *$A(i) = 2^{A(i-1)}$; $A(0) = 1$*
  - ◆ *$A(1) = 2$, $A(2) = 4$, $A(3) = 16$, $A(4) = 2^{16} = 65536$, $A(5) = 2^{65636}$,*
  - ◆ *$A(6)$ = VERY VERY VERY BIG!*
- *Inverse Ackerman:*
  - ◆ *i= Log*(N) = min number times you take $\log_2$ to get equal or smaller than below 1.*
  - ◆ *Worst case Union-by-Rank with path compression is O(M log^*(N) for M unions.*

Binomial Queue

- Combine: Priority Queue and Union/Find: log(N) Forest of trees:

  $H = n_0 B_0 + n_1 B_1 + n_2 B_2 + \boxed{?} + n_p B_p$

  with $n_i = 0,1$

- B_0 = root, $B_{k+1} = B_k + B_k$ attached to root of first. Since B_k has 2^k nodes this is just at binary bit representation N = $(n_p ..., n_0)$ for nodes of size $N \cdot 2^p$

- Build so that min key is in root of B_k.

- Adding H_1 to H_2 is binary arithmetic $O(p = \log(N))$

- Always combing B_k + B_k \rightarrow B_{k+1} with min at root.

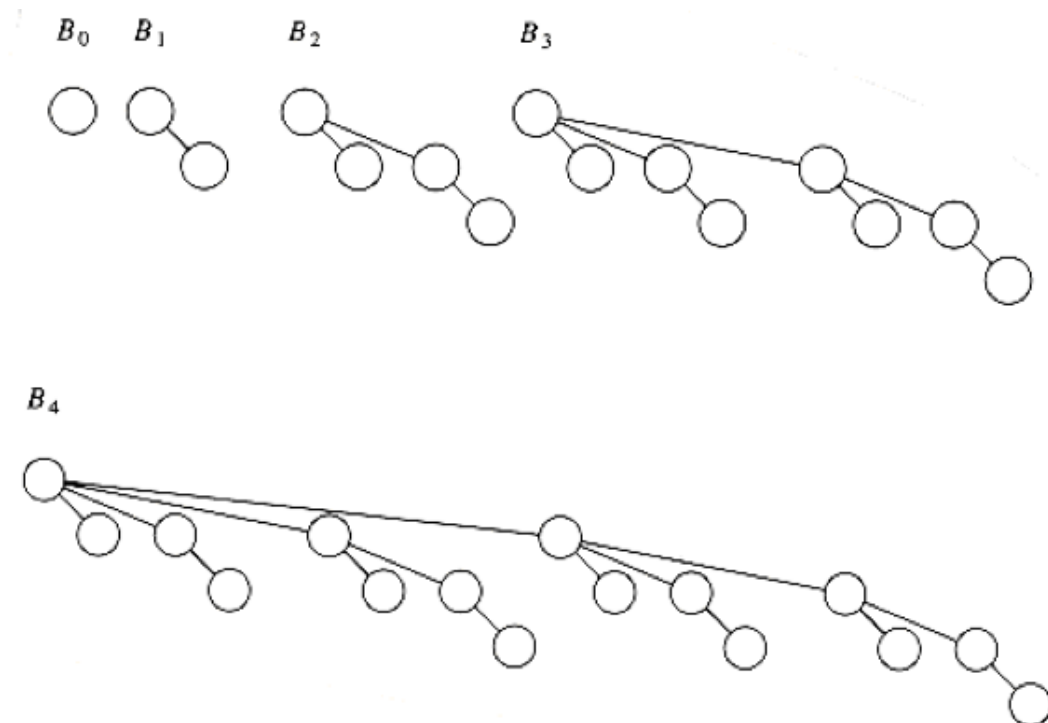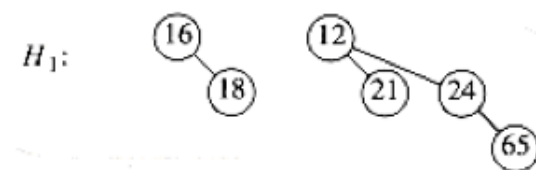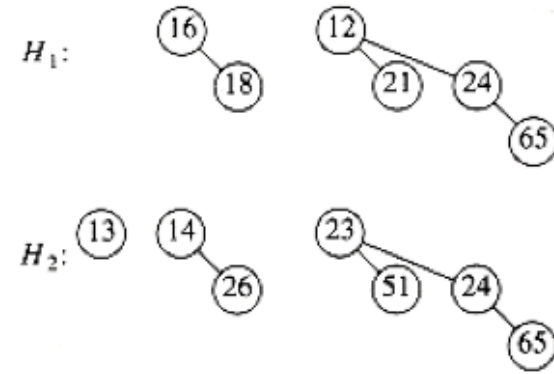Figure 6.34 Binomial trees B_0, B_1, B_2, B_3, and B_4
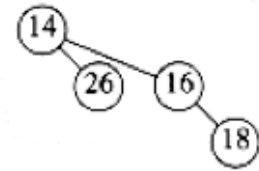
H_1:

Figure 6.36 Two binomial queues H_1 and H_2

Figure 6.37 Merge of the two B_1 trees in H_1 and H_2

23