■設問１．文字列照合

以下の文章を読んで，［ ア ］～［ ク ］にあてはまる適切な語句や記号，数値を，次の選択肢から選んで，それぞれ答えなさい．

テキスト S 中に，単語 W が出現するか否かを判断したい．ここで，各文字列は文字配列として格納されており，S，W の文字長をそれぞれ，|S|，|W| と表すこととする．例えば，S=”string” の場合，S[0]=’s’，S[1]=’t’，|S|=6 のようになるということである．

簡単のため，まずは，テキスト S 中に，文字 X が出現するかを判断する「文字照合問題」を考える．この場合，S の先頭の文字から末尾の文字まで，随時，X との比較を行っていけばよい．すなわち，文字位置 i を 0 から |S| まで 1 ずつ増分していき，随時，S[i]=X となるかを調べるということである．文字 X が出現した場合には，出現した文字位置が返され，出現しなかった場合には，その時点での文字位置である |S| が返されることとなる．よって，例えば，S=”string” に対して，X=’r’ の場合に返される値は［ ア ］であり，X=’a’ の場合に返される値は［ イ ］となる．この処理は，一般的な［ ウ ］探索での処理と同一であることからも分かるとおり，実際の処理量（計算時間）は |S| 程度であり，文字数を n として，この計算量は O(n) で表される．

これを踏まえ，テキスト S 中に，単語 W が出現するかを判断する「文字列照合問題」を考える．単純な手法として，文字照合と同様に S 中の各文字位置において W の先頭文字との比較を行い，一致した場合には，単語 W の先頭文字以降との比較を随時行っていく方法が考えられる．すなわち，$i = 0$ から |S| まで，随時，$j = 0$ として，S[$i + j$]=W[j] となるかを調べ，これが真であった場合，j を 1 ずつ増分していき |W| まで同様に S[$i + j$]=W[j] となるかを調べるということである．$j = |W|$ となった場合には，文字列が一致したことを意味しているので，文字位置 i を返し，処理を終了する．$j < |W|$ において，S[$i + j$]=W[j] とならなかった場合には，文字列が一致しなかったことを意味しているので，i を 1 増分し，同様の処理を繰り返す．このように，各文字位置 i においては，最大で |W| 回の比較が行われることとなる．よって，この単純な文字列照合手法における計算量は O(［ エ ］) で表されるが，実際の処理量（計算時間）は，|S| × |W| 程度になることが予想される．

ここで，W=”abcaba” の文字列照合を行うことを考える．S=”axxxxxxx” の場合，$i = 0$ のとき，$j = 1$ で文字が一致しないことが分かる．よって，次の比較は，$i = 1$，$j = 0$ から行われる．S=”abxxxxxx” の場合，$i = 0$，$j = 2$ で文字が一致しないことが分かる．$j = 1$ では文字が一致していたことが分かっているので，この文字は ’b’ であり，’a’ ではありえない．これを利用して，次の比較は，$i = 1$ から行う必要はなく，$i = 2$，$j = 0$ から行えばよい．同様にして，S=”abcxxxxx” の場合，$i = 0$，$j = 3$ で文字が一致しないことから，次の比較は，$i = 3$，$j = 0$ から行われる．

しかしながら，S=”abcabxxx” の場合には，$i = 0$，$j = 5$ で文字が一致しないからといっ

て，次の比較を $i = 5$ から行うわけにはいかない．なぜなら，$i = 5$ よりも手前の文字列において，一致が発生する可能性があるからである．この文字列の一致を考慮して，次の比較は，$i =$ [オ] から行う必要がある．この際，前回の比較において，$j = 5$ まで文字が一致していたという事実から，次の比較は $j = 0$ から行う必要はなく，$j =$ [カ] から行えば十分であることが分かる．このように，次の比較の際に，単純に i を 1 増やし，$j = 0$ とするだけでなく，単語 W における文字の重複状況に応じて i，j を適切に管理することにより，文字の比較回数を減少できることが期待される．

このような処理を実現する手法の 1 つとして [キ] 法が存在する．この計算量は O([ク]) で表されるが，これにかかる実際の処理量（計算時間）は最大でも 2|S| 程度となることが知られており，単純な手法と比較して，処理を効率化できる可能性がある．

選択肢
グラフ，線形，二分，CYK，KMP，PL，0，1，2，3，4，5，6，$\log n$，$\frac{1}{n}$，$\frac{n}{2}$，n，$2n$，$n \log n$，n^2，2^n，$n!$

■設問２．マージソート

自然マージソートにより，ファイル F に格納されている以下の 13 件のデータを昇順にソートする．

F： 8 2 3 15 34 28 36 13 62 18 71 65 11

[問 1] ファイル F のデータを連（run）で分割した結果を答えなさい．ただし，連の分割はスラッシュ（/）により明示すること．

[問 2] それぞれの連をファイル A, B へ分配，すなわち，交互に格納していく．最初の連をファイル A に格納するものとしたとき，ファイル A, B それぞれにおける，分配した結果を答えなさい．

[問 3] ファイル A, B において，再度，連を作成し，それぞれを順にマージし，ファイル F に格納する．このとき，ファイル F の結果を答えなさい．

[問 4] 同様に，分配，マージを繰り返していき，ソートを行う．このとき，最終的なソート結果を答えなさい．ただし，最終的な結果に至るまでの，分配，マージの結果をすべて示すこと．

■設問３．二分探索

昇順にソートされている以下の 9 件の整数データが配列 a[] に格納されており，これから任意のデータを二分探索により検索することを考える．

a[]： 1 3 31 53 58 61 65 91 95

二分探索は右に示す関数 binsearch() により実現されるものとする．この関数は，検索対象となるデータ x，配列 a[]，その要素数 n=9 を引数として呼び出される．この関数に，次のデータを引数 x として与えた場合，返される値をそれぞれ答えなさい．
（例）53 を引数 x として与えた場合，返される値は 3 となる．

```
int binsearch(int x, int *a, int n)
{
    int m, l=0, r=n-1;

    if (x<=a[l]) return 0;
    if (x>a[r]) return n;
    while (r-l>1) {
        m=(r+l)/2;
        if (x<=a[m]) r=m;
        else l=m;
    }
    return r;
}
```

[問 1]　61

[問 2]　97

[問 3]　10

（下書き用）

要素番号	0	1	2	3	4	5	6	7	8
要素	1	3	31	53	58	61	65	91	95
1 回目									
2 回目									
3 回目									
4 回目									
5 回目									
6 回目									
7 回目									
8 回目									
9 回目									
10 回目									

■設問４．ハッシュ法

ハッシュ法（hashing）により，文字列で表されるキーを，適当な大きさを持つ配列に格納する．キーは互いに重複しないものとする．3 文字以上で構成される文字列 s をキーとして，配列の要素数 N，ハッシュ関数 H，ダブルハッシュ法における別のハッシュ関数 H_i を以下のように定義する．

$$N = 1753 \qquad H(s) = 3 \times s[0] + 7 \times s[2] \qquad H_i(s) = 11 \times s[1] + strlen(s)$$

ここで，$s[0], s[1], s[2]$ は，それぞれ，文字列 s における 1 文字目，2 文字目，3 文字目の文字コードを表しており，$strlen(s)$ は文字列 s の長さ（文字長）を表している．なお，各文字コードは，10 進数で以下のようになっている．

A = 65　B = 66　C = 67　D = 68　...　G = 71　...　O = 79　...　T = 84
...　W = 87　X = 88　Y = 89　Z = 90

例えば，キーとして文字列 ”XYZ” が与えられたとき，その一次インデクス値（primary index value），および，仮にこれが衝突（collision）していた場合に，ダブルハッシュ法において，別のハッシュ関数により計算される増分の値は以下のとおりである．

$$H(\text{”XYZ”}) = 3 \times 88 + 7 \times 90 = 894 \qquad H_i(\text{”XYZ”}) = 11 \times 89 + 3 = 982$$

いま，配列において，以下の要素番号の場所には，すでに，データが格納済みである．

700〜705, 788, 805〜810, 1508, 1675〜1680

ここでは，これを初期状態と呼ぶ．このとき，以下の問いに答えなさい．

[問 1]　初期状態の配列から特定のキーを探索する．

(a) 要素番号 703 の場所に，キー ”DOG” が格納されている．これを線形探査法（linear probing）に従い探索する際，走査することになる要素番号を順番にすべて答えなさい．

(b) 要素番号 1679 の場所に，キー ”BOW” が格納されている．これをダブルハッシュ法（double hashing）に従い探索する際，走査することになる要素番号を順番にすべて答えなさい．

[問 2]　初期状態の配列において，それぞれの手法で，以下のキーを，この順番で格納する．
”BAT”，”BATTER”，”BATTERY”

(a) 線形探査法に従うとき，格納される要素番号をそれぞれ答えなさい．

(b) ダブルハッシュ法に従うとき，格納される要素番号をそれぞれ答えなさい．

■設問５．循環・重連結リスト

図 5-1 の循環・重連結リストに対して，下記ソースコードにおける insert_cdl_list 関数を実行すると，循環・重連結リストは図 5-2 のように変化する．ここで，ソースコードに付加されている番号（①〜⑤）と図 5-2 の番号は対応している．

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;

int insert_cdl_list(int x) {
    node *q, *p =
      (node *)malloc(sizeof(node));
    if(p == NULL)
        return 0;
    p -> num = x;

①  q = start -> left;
②  p -> right = start;
③  q -> right = p;
④  p -> left = q;
⑤  start -> left = p;

    return 1;
}

int insert_left(node *p, int x) {
    node *q =
      (node *)malloc(sizeof(node));
    if(q == NULL)
        return 0;
    q -> num = x;

⑥  q -> right = p;
⑦  p -> left -> right = q;
⑧  q -> left = p -> left;
⑨  p -> left = q;

    return 1;
}
```

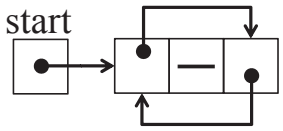


図 5-1

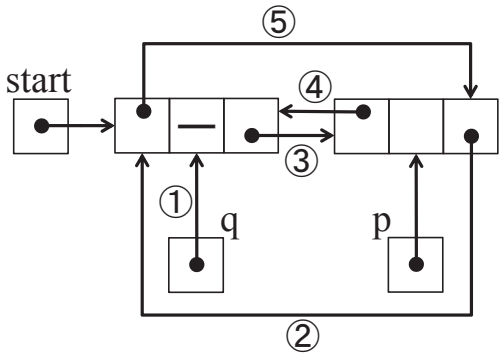


図 5-2

[問 1]　図 5-3 に示す循環･重連結リストに対して，上記ソースコードにおける insert_cdl_list 関数の実行中，③ を実行し終わった時点での途中経過を図示せよ．解答図には，図 5-2 同様，ソースコードに付加されている番号（①〜③）を明記すること．また，番号のつかない矢印も必要に応じて記すこと．

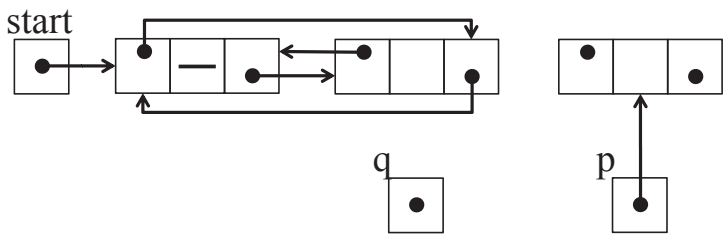


図 5-3

[問 2]　図 5-4 に示す循環・重連結リストに対して，上記ソースコードにおける `insert_left` 関数の実行中，⑧ を実行し終わった時点での途中経過を図示せよ．解答図には，図 5-2 同様，ソースコードに付加されている番号（⑥〜⑧）を明記すること．また，番号のつかない矢印も必要に応じて記すこと．

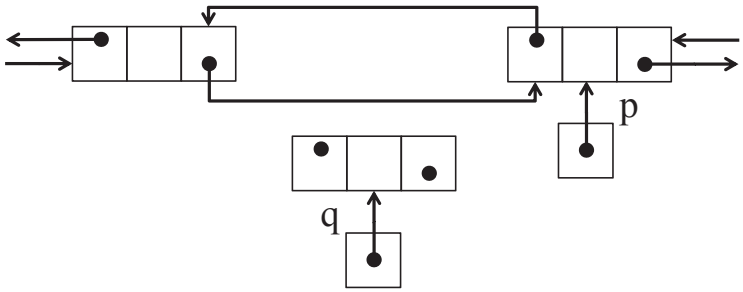


図 5-4

■設問６．二分木のバランス

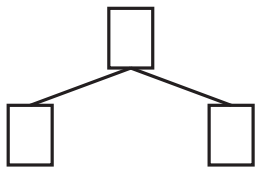
キーボードから n 個の数値を読み込みながら，完全にバランスした二分木を生成するソースコードを以下に示す.

下記 10 個の数値に対して，下記ソースコードの `pbtree` 関数を実行した時の，完全にバランスした二分木を図示せよ．数値は左から順番にキーボードから入力されるものとする．

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;

node *pbtree(int n) {
    int nleft, nright;
    int nleftplusright = n - 1;
    node *p;
    if(n == 0)
        return NULL;
    nleft = nleftplusright / 2;
    nright = nleftplusright - nleft;
    p = (node *)malloc(sizeof(node));
    p -> left = pbtree(nleft);
    scanf("%d", &p -> num);
    p -> right = pbtree(nright);
    return p;
}
```



（下書き用）

■設問７． B-木

以下の条件を満たす次数 2 のB-木に対して，問いに答えよ．

- 各ノードは最大 4 個のデータを含む．
- ルートノード以外の各ノードは少なくとも 2 個のデータを含む．
- ルートノードは少なくとも 1 個のデータを含む．
- 各ノードのデータは昇順に並んでいる．
- 全ての葉は，同じレベルにある．

[問 1]　図 7 に示すB-木に 45 を挿入した後のB-木を図示せよ．

[問 2]　図 7 に示すB-木から 10 を削除した後のB-木を図示せよ．

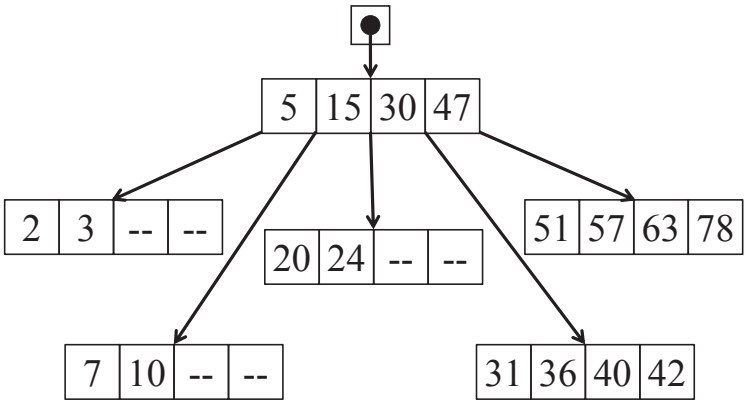


図 7